Clube Delphi +PHP

Ano 8 · Edição 96 · R$9,90

DevMedia

EasyDelphi

Construa e utilize na prática Web Services em suas aplicações ASP.NET

Veja como criar barras de progresso personalizadas para acompanhar o andamento de ações no sistema

DataCar

Orientado a Objetos

Um dos sistemas mais populares desenvolvidos na Revista ClubeDelphi agora totalmente reconstruído usando técnicas de POO

ISSN 1517990-7

NA WEB

Confira no portal ClubeDelphi PLUS 4 vídeo-aulas sobre StarTeam e novidades do Firebird 2.1

Curso de ASP.NET 2.0 e RAD 2007
Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0 - Parte 2

Mini-Curso
Veja como criar um sistema on-line de controle para uma vídeo-locadora, com ASP.NET – Parte 4

PHP
Construa um Web Site completo com PHP e Firebird

Dicas de segurança no Firebird
Saiba como trabalhar com usuários/roles e aplicar ou revogar privilégios de acesso à tabelas e/ou campos

Novidades
Entendendo e implementando RSS em suas aplicações

Coluna: Ask the Expert

developers webdays 2008

100% de aproveitamento!

22 e 23 de agosto – SP

- Conheça as principais novidades das tecnologias **Java e .NET**
- Palestras sobre **Boas práticas e Arquitetura**
- Certificado de Participação
- **100% de aproveitamento** - Apesar das palestras acontecerem simultaneamente, não haverá perda de conteúdo. No dia do evento você receberá um DVD com o conteúdo completo de todas as palestras em formato de vídeo-aula!

São + de 46 horas de conteúdo.

Principais Temas:

Java	.NET	Extras
Ajax	Linq	SOA
Java FX	WPF	WEB 2.0
Grails	Silverlight 2.0	WEB 3.0
Android	ASP.NET 3.5	Arqutetura Web
Groovy	Ajax	Scrum
Jboss Seam	ASP.NET MVC	Microsoft XNA
JSF	Visual Studio 2008	Ruby on Rails
RIA (Flex, etc)	Novidades C# 3.0 e VB.NET 9.0	Domain Driven Design
Best practices em java	Best practices em .NET	Mashups

Maiores informações:
www.devmedia.com.br/webdays2008
evento@devmedia.com.br
(21) 3382-5025

Inscreva-se já!

Realização DevMedia group

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Boas Práticas
Mão na Massa
Mini-Curso e Expert

08 – Sistema DataCar em OO
Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 1
[*Paulo Quicoli*]

Mini-Curso
Novidades
Web

18 – Curso de ASP.NET 2.0 e RAD 2007
Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0 - Parte 2
[*Adriano Santos e Guinther Pauli*]

Novidades

24 – RSS no RAD Studio 2007
Entendendo e implementando RSS em suas aplicações
[*Manoel Edésio*]

Mão na Massa
Mini-Curso e Web

30 – Controle on-line de vídeo-locadora - Parte 4
Veja como criar um sistema on-line de controle para uma vídeo-locadora
[*Maikel Marcelo Scheid*]

Banco de Dados

36 – Dicas de segurança no Firebird
Saiba como trabalhar com usuários/roles e aplicar ou revogar privilégios de acesso à tabelas e/ou campos
[*Adriano Santos*]

PHP

44 – Desenvolvendo Web Sites com PHP e Firebird
Veja como utilizar o banco de dados Firebird em aplicações PHP
[*Adriano Santos*]

Easy Delphi

56 – Iniciando o uso de Web Services no .NET
Construa e utilize na prática Web Services em suas aplicações ASP.NET
[*Maikel Marcelo Scheid*]

Easy Delphi

62 – Progresso em ações do sistema
Veja como criar barras de progresso personalizadas para acompanhar o andamento de ações no sistema
[*Maikel Marcelo Scheid*]

Você percebeu os nomes ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Banco de dados] Artigo sobre o universo de persistência de dados.

[Mini-Curso] Artigo que faz parte de um mini-curso.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[PHP] Artigo com foco na linguagem PHP (não necessariamente utilizando o Delphi PHP).

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre.

[Boas Práticas] A Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas, mas também as melhores práticas – Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

ClubeDelphi

Ano 8 - 96ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Adolfo Sabino
simininu@yahoo.com.br

Capa
Antonio Xavier
webdesigner@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

Apoio
ASSESPRO PARANÁ

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

EDITORIAL

Na edição 29 da Revista ClubeDelphi, há quase 6 anos, convidei o amigo Fernando Prass para então construirmos um sistema a ser publicado em uma série de artigos. Na época, Delphi 6, o objetivo era demonstrar o que havia de mais novo em termos de tecnologia. CLX, dbExpress e Firebird eram (e muitos ainda são) os pontos em alta para os desenvolvedores. Ao mesmo tempo, queríamos um sistema original, não trivial, que não fosse algo do tipo "controle de clientes". Estava nascendo o DataCar, um dos mini-cursos mais populares em toda a história da Revista. Você pode baixá-lo por completo no meu endereço na CC, a partir do link "Guinther Pauli" em http://cc.codegear.com/PopularAuthors.aspx.

Muita coisa surgiu de lá para cá. .NET, Web, AJAX, WPF, Silverlight, LINQ, Windows Vista, e a idéia de reconstruir o DataCar era eminente. Pensamos logo em duas opções: DataCar OO e DataCar Web. Nesta edição iniciamos a primeira opção, onde Paulo Quicoli começará a reconstruir o DataCar do zero, porém fazendo total e completo uso de poderosas técnicas de POO (Programação Orientada a Objetos) e é claro, muita UML. Confira esse imperdível mini-curso e comece já a desenvolver seus aplicativos de uma forma mais robusta, usando boas práticas de programação e com o que há de novo em tecnologia.

Ainda nesta edição, na companhia do Adriano, continuo a série que mostra um pequeno sistema para você dar os primeiros passos nesse fabuloso mundo que é o ASP.NET 2.0, suportado agora pelo RAD Studio 2007. Acredite, o ASP.NET 2.0 torna o desenvolvimento de aplicações Web tão RAD quanto é o desenvolvimento com a VCL. Então, não existem mais desculpas para deixar aquele antigo sonho de criar uma versão Web do seu projeto Desktop. A hora é agora. E vem mais por aí: AJAX no ASP.NET (sem falar no AJAX Control Tool Kit).

Ainda sobre o RAD 2007, o Manoel Edésio mostra como trabalhar com RSS no Delphi, esse padrão que permite "que as notícias venham até você, e não o contrário". Mantenha-se informado do que há de mais novo em tecnologia criando o seu próprio leitor RSS, ao mesmo tempo que aprende mais sobre a tecnologia.

Agora falando em ASP.NET 1.1, o Maikel continua o seu mini-curso de vídeo-locadora, falando sobre usuários e sessões. O Adriano também fala sobre controle de usuários, só que no Firebird, importante fundamento para garantir a segurança do banco de dados.

Na sessão Easy Delphi, para os iniciantes, temos dois artigos: aprenda a criar seus próprios Web Services (incluindo servidores) e como utilizar o ProgressBar, Gauge e TrackBar para indicar o progresso de ações no sistema. Para finalizar, o Adriano nos traz um excelente artigo sobre PHP.

Grande abraço e sucesso com o Delphi (e com o PHP também).

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 600 Vídeo Aulas!
- 7 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as seguintes informações: **Login: DVM.PL e Senha: MKI810**

O acesso é válido por 30 dias a partir da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Assinatura

ClubeDelphi PLUS

Mais conteúdo Delphi por menos

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+600 vídeo aulas e 7 cursos online

Brinde na web desta edição

3 Vídeos

Novidades do Firebird 2.1
www.devmedia.com.br/articles/viewcomp.asp?comp=8799

Impressão de notas fiscais em impressões matriciais com RDPrint
www.devmedia.com.br/articles/viewcomp.asp?comp=8597

StarTeam – Curso básico I
www.devmedia.com.br/articles/viewcomp.asp?comp=8396

StarTeam – Curso básico II
www.devmedia.com.br/articles/viewcomp.asp?comp=8493

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link: *www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:

Login: **DVM.PL** Senha: **KMI810**

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Criando colunas em um ComboBox

Adriano, gostaria de saber como fazer para mostrar um *ComboBox* em colunas, é possível?

Pedro

Oi Pedro, sim podemos colocar colunas em um *ComboBox*. É um pouco trabalhoso, mas nada que algumas linhas de programação não resolvam.

Primeiramente inclua um *ComboBox* e um *Button* em um formulário. No evento *OnDrawItem* do *ComboBox* digite o código da **Listagem 1**. Em seguida adicione a linha a seguir no *OnCreate* do formulário ou se preferir configure diretamente na propriedade *Style* do *ComboBox* em tempo de projeto.

```
Combobox1.Style := csOwnerDrawFixed;
```

Por fim, inclua o código da **Listagem 2** no *OnClick* do *Button*. Rode a aplicação, clique no botão e veja o resultado no *ComboBox*. (**Figura 1**)

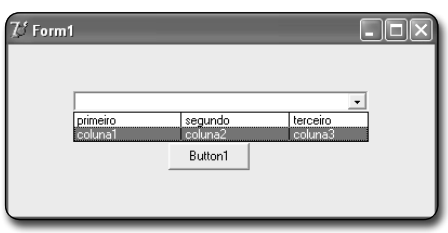


Figura 1. Exemplo de ComboBox com colunas

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 1. Draw Item do ComboBox

```
procedure TForm1.ComboBox1DrawItem(Control:
  TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);

var
  strVal, strAll: string;
  pos1, pos2: Integer;
  rc: TRect;
  arrWidth: array[0..3] of Integer;

begin
  Combobox1.Canvas.Brush.Style := bsSolid;
  Combobox1.Canvas.FillRect(Rect);
  strAll := Combobox1.Items[Index];

  arrWidth[0] := 0;
  arrWidth[1] := 100;
  arrWidth[2] := 200;
  arrWidth[3] := 300;

  rc.Left := Rect.Left + arrWidth[0] + 2;
  rc.Right := Rect.Left + arrWidth[1] - 2;
  rc.Top := Rect.Top;
  rc.Bottom := Rect.Bottom;

  pos1 := Pos(';', strAll);
  strVal := Copy(strAll, 1, pos1 - 1);
  Combobox1.Canvas.TextRect(rc, rc.Left, rc.Top, strVal);
  Combobox1.Canvas.MoveTo(rc.Right, rc.Top);
  Combobox1.Canvas.LineTo(rc.Right, rc.Bottom);

  rc.Left := Rect.Left + arrWidth[1] + 2;
  rc.Right := Rect.Left + arrWidth[2] - 2;

  strAll := Copy(strAll, pos1 + 1, Length(strAll) - pos1);
  pos1 := Pos(';', strAll);
  strVal := Copy(strAll, 1, pos1 - 1);

  Combobox1.Canvas.TextRect(rc, rc.Left, rc.Top, strVal);
  Combobox1.Canvas.MoveTo(rc.Right, rc.Top);
  Combobox1.Canvas.LineTo(rc.Right, rc.Bottom);

  rc.Left := Rect.Left + arrWidth[2] + 2;
  rc.Right := Rect.Left + arrWidth[3] - 2;

  strAll := Copy(strAll, pos1 + 1, Length(strAll)
    - pos1);
  pos1 := Pos(';', strAll);
  strVal := Copy(strAll, 1, pos1 - 1);

  Combobox1.Canvas.TextRect(rc, rc.Left, rc.Top, strVal);
  Combobox1.Canvas.MoveTo(rc.Right, rc.Top);
  Combobox1.Canvas.LineTo(rc.Right, rc.Bottom);
  strAll := Copy(strAll, pos1 + 1, Length(strAll) - pos1);

end;
```

Listagem 2. Códigos dos métodos para interagir com o mundo externo

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Combobox1.Items do
  begin
    Add('primeiro;segundo;terceiro;');
    Add('coluna1;coluna2;coluna3;');
  end;
end;
```

POR QUE A ALOG É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB ACCREDITED BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br I 0800 282 3330

Hosting Gerenciado® I Colocation I Contingência I Email Corporativo I Conectividade I Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Sistema DataCar em OO

Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 1

Nesse artigo veremos

- Regras de Negócio;
- UML;
- Diagrama de Classes;
- Casos de Uso;
- Testes unitários;

Qual a finalidade

- Criar um software seguindo os padrões atuais da engenharia de software e OO, adotando boas práticas de desenvolvimento;

Quais situações utilizam esses recursos?

- Sistemas com manutenção efetiva, que necessitam estar preparados para mudanças de regras de negócio e exigem uma modelagem prévia;

Paulo Roberto Quicoli
(pauloquicoli@gmail.com)
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente/servidor. Formado em Tecnologia de Processamento de dados pela FATEC, na cidade de Taquaritinga/SP

Desde o início da ClubeDelphi, o intuito foi ensinar e compartilhar conhecimento com a comunidade Delphi de uma forma totalmente diferenciada. Uma forma que atendesse os anseios dos desenvolvedores sempre falando da maneira que todos entendessem melhor. Com esse objetivo em mente viu-se que a melhor forma de transmitir esse conhecimento é através do desenvolvimento de exemplos e assim tem sido até hoje. Em outra época quando a novidade da vez era o DBExpress no então Delphi 6, nossos colegas Guinther Pauli e Fernando Prass se uniram para desenvolver um dos artigos mais populares da comunidade, que demonstraria como projetar e codificar usando o que havia de melhor, direcionando assim desde o programador mais jovem ao mais experiente. Nascia assim o projeto do sistema DataCar que gerencia revendas de automóveis, no popular, garagens.

Hoje, passados alguns anos, novas tecnologias surgiram e se tornaram populares como por exemplo o .NET. No mundo web então nem se fala, ASP.NET, AJAX, Ruby, PHP, Silverlight. Programação orientada a objetos é um dos assuntos de mais destaque na comunidade, sem falar de seus "acessórios" como MVC, Padrões de Projeto, Testes unitários, UML. As ferramentas também

Metodologia

O exemplo deste artigo utiliza a arquitetura OO para criação do sistema. Utiliza também a UML para modelagem do projeto e DAO para persistência dos dados, além de ser um sistema multicamadas com mapeamento objeto relacional. O sistema será multicamadas porque fará uso do MVC (Model View Controller). Para mais informações sobre essa arquitetura e outras, visite http://www.devmedia.com.br/articles/viewcomp.asp?comp=5219

evoluíram e muito. Nessa trajetória chegamos ao Delphi 2007 pra win32 e RAD Studio 2007 que envolve Win32 e .NET.

Como o objetivo principal da ClubeDelphi não mudou, quero com muito orgulho trazer de volta o projeto DataCar, também desta vez utilizando, como no projeto original, o que há de melhor. Neste projeto vamos utilizar Delphi 2006, mas sinta-se livre para utilizar a versão que possuir, e aplicaremos ao projeto e programação orientados a objetos, abordando testes unitários, mapeamento objeto relacional, separação de camadas, persistência e relatórios.

Nota: Para fins didáticos e desenvolvimento dos diagramas de caso de uso e de classe vamos utilizar o ModelMaker 9. Na minha opinião é o melhor para se usar com Delphi. Quanto aos testes unitários vamos utilizar o DUnit. Ambos podem ser baixados, respectivamente, nos endereços *www.modelmakertools.com* e *dunit.sourceforge.net*. No entanto, não vamos criar todos os diagramas da UML do zero, pois isso exigiria uma revista inteira. Nem mostraremos passo a passo como criar esses diagramas, pois não é o objetivo do artigo. Você pode facilmente observar os diagramas propostos e criá-los do MM. Para mais informações sobre o MM veja as edições 40 e 56 da Revista ClubeDelphi.

Entendendo as regras de negócio

No geral, de forma simplista, uma pessoa pode entrar em contato com a garagem para informar seu interesse em comprar determinado modelo de carro, até um valor específico para que posteriormente seja notificada sobre a disponibilidade. Essa pessoa se torna um cliente da garagem e com isso seus dados pessoais são registrados tais como nome, endereço, telefones e documentos. Também uma pessoa pode deixar na garagem seu carro para ser vendido por um determinado valor e então cabe à garagem tentar realizar essa venda. Essa pessoa também se torna um cliente da garagem. Por sua vez um veículo possui modelo que possui uma marca. Também possui cor, acessórios, ano de fabricação/modelo, consome um determinado tipo de combustível, possui um número determinado de portas, uma placa e um proprietário.

Enquanto de posse de um veículo ou quando ela mesma adquire um para posterior venda, a garagem pode ter despesas com o mesmo, por exemplo despesas com documentação, mecânica e auto-elétrica. Ao intermediar uma venda ou vender um veículo próprio, um lucro é esperado e todas essas despesas são contabilizadas para que juntamente definam o valor real da venda.

Modelagem de negócio

No projeto orientado a objetos utilizamos a UML que nos oferece diagramas para representar da melhor forma

Nota do DevMan

A Unified Modeling Language (UML) é uma linguagem de modelagem não proprietária de terceira geração. A UML não é uma metodologia de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como projetar seu sistema, mas ela lhe auxilia a visualizar seu desenho e a comunicação entre objetos.

Basicamente, a UML permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, semântica. É uma notação independente de processos, embora o RUP (Rational Unified Process) tenha sido especificamente desenvolvido utilizando a UML.

É importante distinguir entre um modelo UML e um diagrama (ou conjunto de diagramas) de UML----o último é uma representação gráfica da informação do primeiro, mas o primeiro pode existir independentemente.

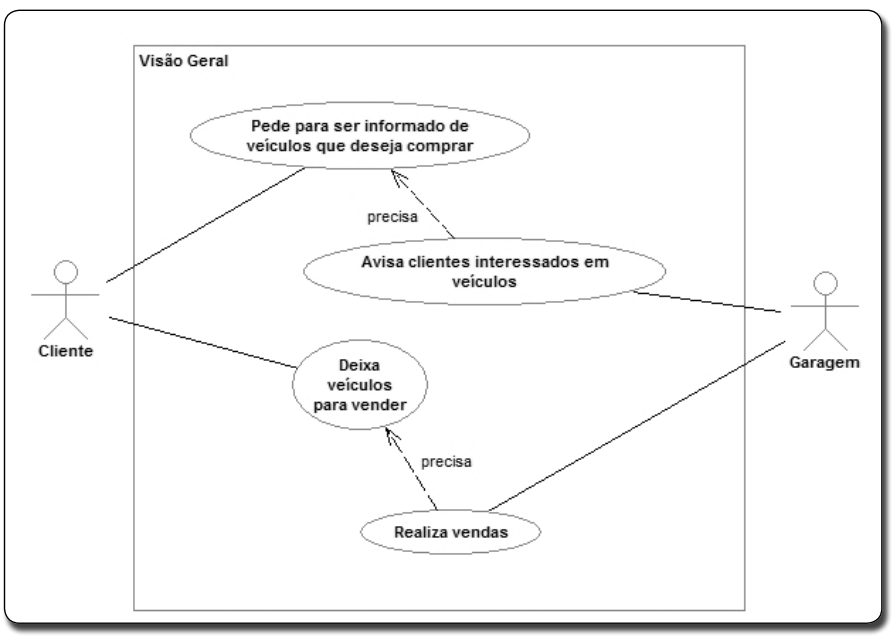


Figura 1. Diagrama de Caso de uso, Visão Geral

possível aquilo que vamos implementar na forma de sistema computacional. Um desses diagramas é o Caso de Uso. Veja o diagrama de caso de uso, na **Figura 1**, que representa uma visão geral desse funcionamento.

O diagrama de caso de uso é uma representação não técnica do negócio envolvido. Ele é a visão do usuário. Um diagrama de caso de uso possui essencialmente atores e suas ações. Os atores representam qualquer coisa que interaja externamente com o sistema. Eles são representados por "homenzinhos". As ações por sua vez são representadas por balões. Essas ações são "explodidas" em outros casos de uso que detalham seu funcionamento. Junto a esses diagramas temos sua descrição.

Descrição de caso de uso

A descrição de um caso de uso por sua vez relata a forma com que o usuário deseja que uma determinada ação seja realizada. Existem várias formas de se descrever um caso de uso, todas similares, porém vou apresentar a de minha preferência. Ao obter as informações do usuário é necessário organizá-las de forma racional, e para isso podemos usar o modelo da **Figura 2.**

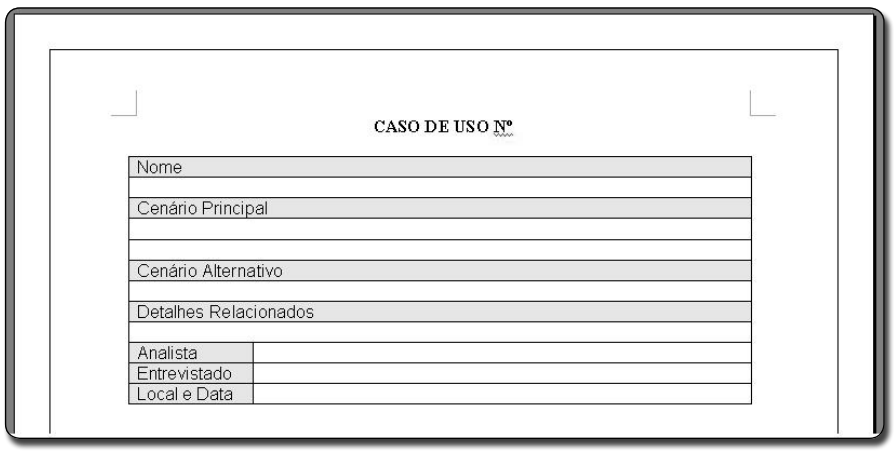
Na **Figura 3,** temos um exemplo de descrição de caso de uso para a ação "Deixa veículos para vender".

Observe que no *Cenário Principal* descrevemos de forma específica as tarefas envolvidas para se cumprir a ação, todas enumeradas na seqüência que o cliente deseja. Não foram envolvidos termos técnicos porque um caso de uso sempre detalha a visão do usuário. Também não são mencionadas exceções, isto porque o cenário principal representa a situação perfeita.

A seção *Cenário Alternativo* é utilizada para realizar a descrição das exceções referentes a cada passo do *Cenário Principal*. As exceções possuem uma numeração constituída da referência do passo a que se aplica mais sua ordem dentro daquele passo.

No *Passo 1* do *Cenário Principal* fazemos uma referência ao caso de uso *Cadastrar Cliente* que ainda não foi desenvolvido, porém já indicamos a necessidade dele aqui. A seção detalhes relacionados pode conter a lógica do negócio envolvida indicando situações especiais que o cliente enfrenta e que devem ser tratadas pelo sistema.

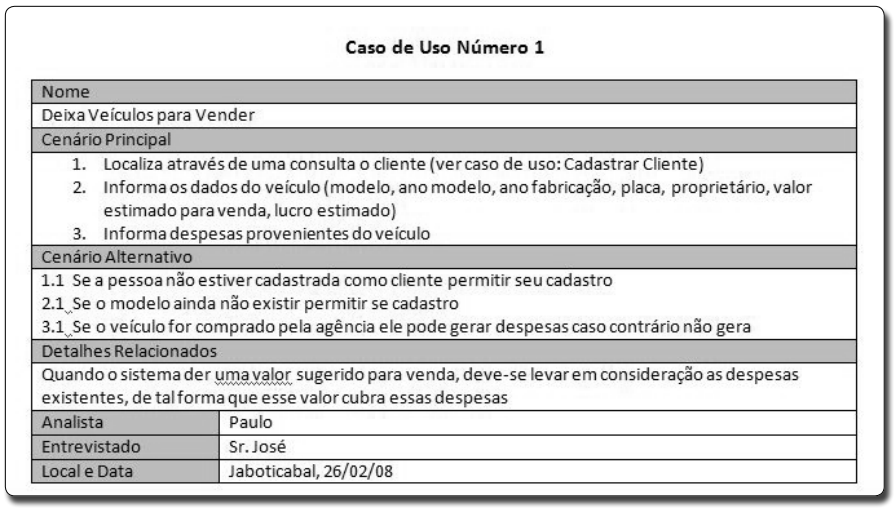
Nem sempre, senão na maioria das vezes, o cliente não sabe como quer que seu sistema funcione. Nessas situações é muito importante dar sugestões baseadas em suas experiências, mostrando até mesmo através de imagens como que determinada ação pode ser implementada ou até mesmo aprimorar os passos que foram definidos, lembrando sempre de se evitar termos técnicos se o cliente for leigo, aplicando assim as regras básicas de comunicação.



CASO DE USO Nº

Nome	
Cenário Principal	
Cenário Alternativo	
Detalhes Relacionados	
Analista	
Entrevistado	
Local e Data	

Figura 2. Modelo de descrição de caso de uso



Caso de Uso Número 1

Nome	
Deixa Veículos para Vender	
Cenário Principal	
1. Localiza através de uma consulta o cliente (ver caso de uso: Cadastrar Cliente) 2. Informa os dados do veículo (modelo, ano modelo, ano fabricação, placa, proprietário, valor estimado para venda, lucro estimado) 3. Informa despesas provenientes do veículo	
Cenário Alternativo	
1.1 Se a pessoa não estiver cadastrada como cliente permitir seu cadastro 2.1 Se o modelo ainda não existir permitir se cadastro 3.1 Se o veículo for comprado pela agência ele pode gerar despesas caso contrário não gera	
Detalhes Relacionados	
Quando o sistema der uma valor sugerido para venda, deve-se levar em consideração as despesas existentes, de tal forma que esse valor cubra essas despesas	
Analista	Paulo
Entrevistado	Sr. José
Local e Data	Jaboticabal, 26/02/08

Figura 3. Descrição de caso de uso

Diagrama de classes

Baseando-se nessas informações podemos criar um modelo de classes que representa as entidades envolvidas. Essas entidades são as classes que representam diretamente o que poderá ser salvo, recuperado, atualizado e excluído. Na **Figura 4** temos o diagrama de classes.

O ModelMaker pode gerar para nós

todo o código das classes modeladas. Neste exemplo as classes são geradas na *Unit ModelU.pas*, que está adicionada a um projeto do tipo Win32. Não entrarei em detalhe de como utilizar o ModelMaker para construir o diagrama, pois vai além do escopo deste artigo e já foram publicados aqui na revista vários artigos sobre isso.

Porém para um bom entendimento de um diagrama de classes é necessário entender sua simbologia que é definida pela UML, portanto igual em qualquer ferramenta de modelagem.

Basicamente todo diagrama de classes é constituído por classes, suas associações, agregações e composições. Uma classe é representada por um quadro que pode conter os nomes de seus atributos (propriedades e campos), métodos e eventos (**Figura 5**).

As classes por sua vez podem possuir

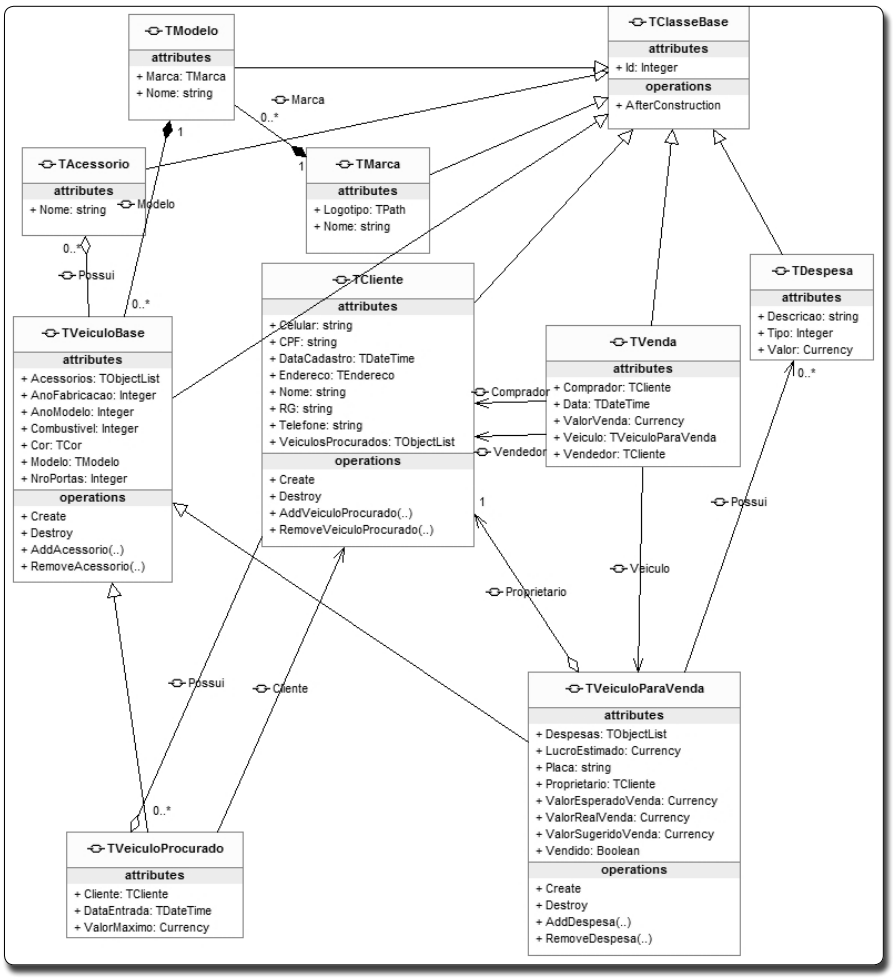


Figura 4. Diagrama de classes

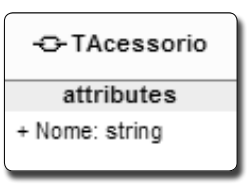


Figura 5. Classe

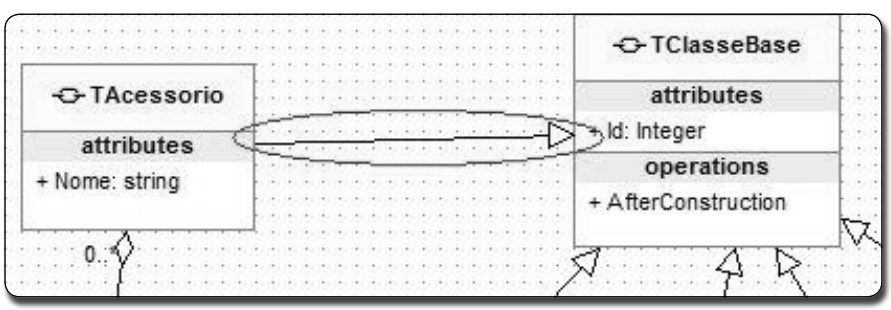


Figura 6. Herança

uma herança vinda de outra classe. A herança é representada por uma linha ligando as classes e na classe a ser herdada a ponta dessa linha possui uma seta (**Figura 6**). Observe em nosso exemplo que todas as classes herdam de *TClasseBase*.

Os relacionamentos entre classes se dão por meio de associações, agregações e composições. Esses relacionamentos possuem multiplicidade. A multiplicidade representa a quantidade mínima e máxima de objetos exigida para existir o relacionamento. Uma associação é um relacionamento estrutural entre instâncias e especifica que objetos de uma classe estão ligados a objetos de outras classes e são representados conforme **Figura 7.** Observe em nosso exemplo que a classe *TVenda* possui uma referência à classe *TCliente* (pois toda vende exige um cliente, nesse caso um Vendedor, que é o atributo em *TVenda*). Imagine isso com uma *foreign-key* em um banco de dados relacional.

As agregações e composições representam relacionamentos do tipo "todo/parte". Na agregação, a classe que exige o relacionamento vamos chamá-la de *A*. E a que contém a classe relacionada vamos chamá-la de *B*. Porém a classe *B* não depende da classe *A* para existir. Veja sua representação na **Figura 8**. Em nosso exemplo, a classe *TVeiculoBase* possui um coleção de objetos do tipo *TAcessorio* (ou seja, veículos têm acessórios).

Na composição, no entanto, temos a classe parte que se relaciona à classe todo e não pode existir sem a classe todo. Um exemplo que vem ajudar no entendimento é *Nota Fiscal* e seus itens. Representando a situação em classes, teríamos a classe *Nota Fiscal* que é o nosso todo. Conseqüentemente temos a classe *Itens* que só existe se a *Nota Fiscal* existir. Isso é uma composição. Sua representação se dá por um diamante preenchido no lado todo, veja **Figura 9.** Em nosso exemplo, um *TModelo* deve possuir uma marca (*TMarca*).

Entendendo as classes criadas

Ao observar o diagrama de classes da **Figura 4**, vemos que todas nossas classes iniciais herdam de *TClassebase* diretamente ou indiretamente. Ela possui o atributo que é comum para todas as entidades criadas, a propriedade *ID*. O *ID* em um objeto indica sua unicidade dentro de todos os outros objetos. O *ID* indica que um objeto é único (imagine isso com a chave-primária de uma tabela). Veja que criamos também uma classe base para veículo, *TVeiculoBase*, que contém os atributos comuns para os outros tipos de veículos que foram definidos, *TVeiculoProcurado* e *TVeiculo ParaVenda*. Inclusive seus relacionamentos serão herdados.

Na **Listagem 1** temos o código fonte da *Unit ModelU.pas*, contendo algumas das classes até então criadas. A *Unit* completa está disponível para download.

> **Nota**: Por restrições de espaço, a implementação da **Listagem 1** foi reduzida e encontra-se em sua versão completa no endereço para download deste artigo.

Com nossas classes de negócio implementadas podemos já aplicar alguns testes para identificarmos qualquer bug. Aplicaremos aqui os testes unitários.

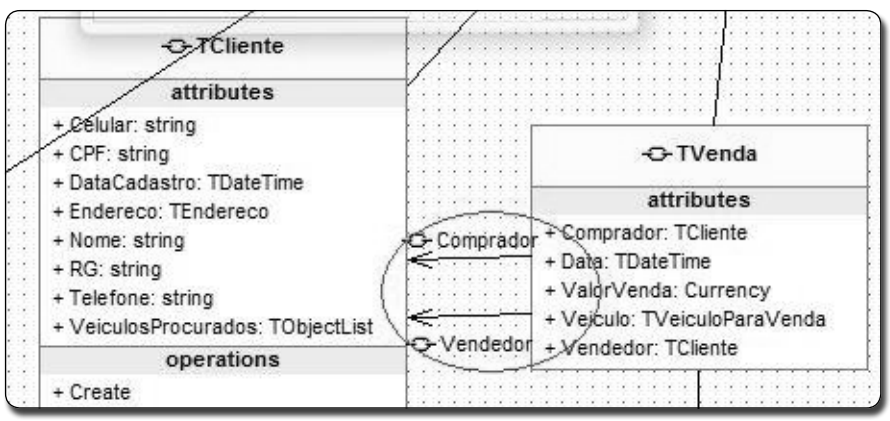


Figura 7. Associação

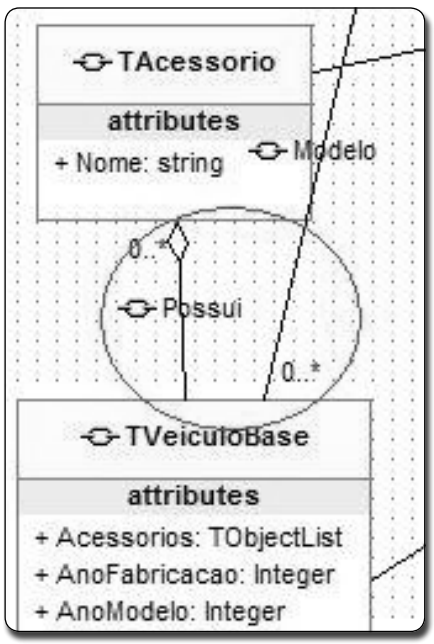


Figura 8. Agregação

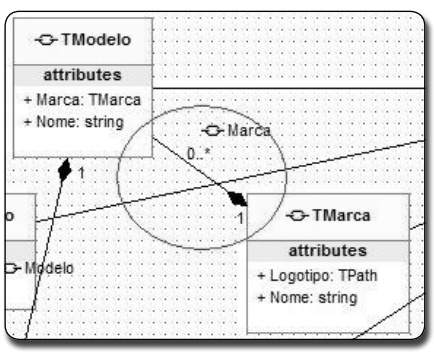


Figura 9. Composição

O que são testes unitários?

Quando se desenvolve orientado a objetos, ou, mesmo quando seu sistema não é OO no todo, apenas algumas partes dele, é possível testar essas classes de forma automatizada, que é algo que o desenvolvedor Delphi não está habituado a fazer no seu dia-a-dia. Esses testes são aplicados nas classes, individualmente com o objetivo de detectar algum mau funcionamento nela e não no sistema todo. Claro que se seu sistema for todo OO praticamente todo o sistema será testado.

A principal vantagem que vejo nesse tipo de teste é a segurança obtida para se realizar qualquer alteração no sistema. Imagine a situação onde você altera o tipo de uma variável de *Integer* para *Byte*. Em um sistema pequeno você consegue medir, de cabeça mesmo, o impacto dessa alteração, mas em um sistema médio e grande e, principalmente se você não o desenvolve sozinho, é muito difícil se lembrar onde essa simples mudança pode apresentar problemas. E muitas vezes nessas situações o problema acaba sendo identificado pelo cliente e não por você.

É nessa hora que os testes unitários entram pra nos salvar desses bugs. Aplicando os testes a falha seria descoberta automaticamente e corrigida antes de se entregar ao cliente, ou seja, utilizar um Framework de testes unitários nos dá segurança para realizar alterações no sistema em desenvolvimento. Para que se entenda melhor como funciona o processo, vamos aplicar alguns testes em nossas classes.

Utilizando o DUnit

Para Delphi Win32 temos o *Framework DUnit* que pode ser baixado em *dunit.sourceforge.net*. O pacote não requer instalação, apenas que se adicione no *Library Path* sua pasta *Src*, a pasta onde o código fonte de nossa classe está alocada.

O Delphi 2006 nos oferece um assistente para criação de testes, para isso criaremos um projeto e adicionaremos a *Unit ModelU.pas*. Crie uma nova aplicação usando o menu File>New>Vcl Forms Applications – Delphi for Win32 e salve-o como "DataCar2008.bdsproj".

Listagem 1. Unit ModelU.pas

```
unit ModelU;

interface

uses Classes, SysUtils, Contnrs;

type
  TPath = string;

  TVeiculoBase = class;
  TDespesa = class;
  TCor = class;
  TCidade = class;
  TMarca = class;
  TModelo = class;
  TAcessorio = class;
  TEndereco = class;
  TCliente = class;
  TVeiculoProcurado = class;
  TVeiculoParaVenda = class;
  TVenda = class;

  TClasseBase = class(TObject)
  private
    FId: Integer;
    procedure SetId(const Value: Integer);
  public
    procedure AfterConstruction; override;
    property Id: Integer read FId write SetId;
  end;

  TVeiculoBase = class(TClasseBase)
  private
    FAcessorios: TObjectList;
    FAnoFabricacao: Integer;
    FAnoModelo: Integer;
    FCombustivel: Integer;
    FCor: TCor;
    FModelo: TModelo;
    FNroPortas: Integer;
    procedure SetAnoFabricacao(const Value: Integer);
    procedure SetAnoModelo(const Value: Integer);
    procedure SetCombustivel(const Value: Integer);
    procedure SetCor(const Value: TCor);
    procedure SetModelo(const Value: TModelo);
    procedure SetNroPortas(const Value: Integer);
  public
    constructor Create; reintroduce;
    destructor Destroy; override;
    procedure AddAcessorio(acessorio: TAcessorio);
    procedure RemoveAcessorio(acessorio: TAcessorio);
    property Acessorios: TObjectList read FAcessorios
      write FAcessorios;
    property AnoFabricacao: Integer read FAnoFabricacao
      write SetAnoFabricacao;
    property AnoModelo: Integer read FAnoModelo
      write SetAnoModelo;
    property Combustivel: Integer read FCombustivel
      write SetCombustivel;
    property Cor: TCor read FCor write SetCor;
    property Modelo: TModelo read FModelo write SetModelo;
    property NroPortas: Integer read FNroPortas
      write SetNroPortas;
  end;

  TVeiculoParaVenda = class(TVeiculoBase)
  private
    FDespesas: TObjectList;
    FLucroEstimado: Currency;
    FPlaca: string;
    FProprietario: TCliente;
    FValorEsperadoVenda: Currency;
    FValorRealVenda: Currency;
    FVendido: Boolean;
    function GetValorSugeridoVenda: Currency;
  public
    constructor Create; reintroduce;
    destructor Destroy; override;
    procedure AddDespesa(despesa: TDespesa);
    procedure RemoveDespesa(despesa: TDespesa);
    property Despesas: TObjectList read FDespesas write FDespesas;
    property LucroEstimado: Currency read FLucroEstimado write FLucroEstimado;
    property Placa: string read FPlaca write FPlaca;
    property Proprietario: TCliente read FProprietario write FProprietario;
    property ValorEsperadoVenda: Currency read FValorEsperadoVenda write FValorEsperadoVenda;
    property ValorRealVenda: Currency read FValorRealVenda write FValorRealVenda;
    property ValorSugeridoVenda: Currency read GetValorSugeridoVenda;
    property Vendido: Boolean read FVendido write FVendido;
  end;
```

Em seguida vá ao Windows Explorer e remova o arquivo *Unit1.pas* do diretório do projeto, pois não vamos usá-lo. Por fim copie o arquivo *UModel.pas*, que acompanha os arquivos de exemplo desse artigo, diretamente no local onde salvou o *.bdsproj*. Em seguida clique em *Add file to Project* ou pressione *Shift + F11* e adicione a *Unit* mencionada anteriormente.

No *Project Manager*, clique com o botão direito no *ProjectGroup1* e escolha *Add New Project*. A tela de projetos irá aparecer e nela teremos o item *Unit Test*. Clique nele e escolha *Test Project*. Pressione o *Ok* e o assistente será aberto, veja **Figura 10**.

Perceba que o assistente já sugere um nome de artigo. O assistente adicionará ao *Project Group* um novo projeto com o nome que especificarmos no campo *Project Name*. Por padrão o nome sugerido é o nome do projeto seguido da palavra *Tests*. A localização do projeto também é sugerida como um subpasta *Test* dentro da pasta do projeto. Vamos manter essas configurações e clicar em *Next* e depois em *Finish*. O último passo do assistente permite escolher o Framework de testes e como ele será executado. O *DUnit* permite que os testes sejam executados e visualizados a partir de uma janela Windows ou a partir do console (*prompt* de comando). Ao terminar o assistente, no *Project manager* está como na **Figura 11**.

O projeto que o assistente criou já configura o ambiente inicial exigido pelo *DUnit*. Agora para criar os testes, basicamente, para cada classe que desejamos testar outra deve ser criada. Por exemplo, se vamos testar a classe *TClasseBase* será preciso criar uma classe *TClasseBaseTest*. Vamos fazer isso agora.

No projeto *DataCar2008Tests* clique com o botão direito sobre ele e escolha *Add New>Unit* e salve-a como "ClasseBaseTest.pas". Nessa *Unit* vamos adicionar duas referências, uma ao *Framework DUnit* e outra referente às classes que vamos testar, portanto temos que adicionar um *Uses* logo abaixo do interface referenciando *TestFramework* e *ModelU*. Se *ModelU* não é localizado pelo Delphi, basta adicionar ao projeto de teste sua localização e incluí-lo também no projeto, veja **Figura 12.**

Criando um TestCase

Como já foi mencionado, o teste nada mais é do que uma classe que testa outra classe. Essa classe teste deve herdar de *TTestCase*. Além disso, podemos sobrecarregar dois métodos protegidos chamados *Setup* e *TearDown*. Os métodos da classe, que são os testes em si, devem ficar na seção *published*. Com

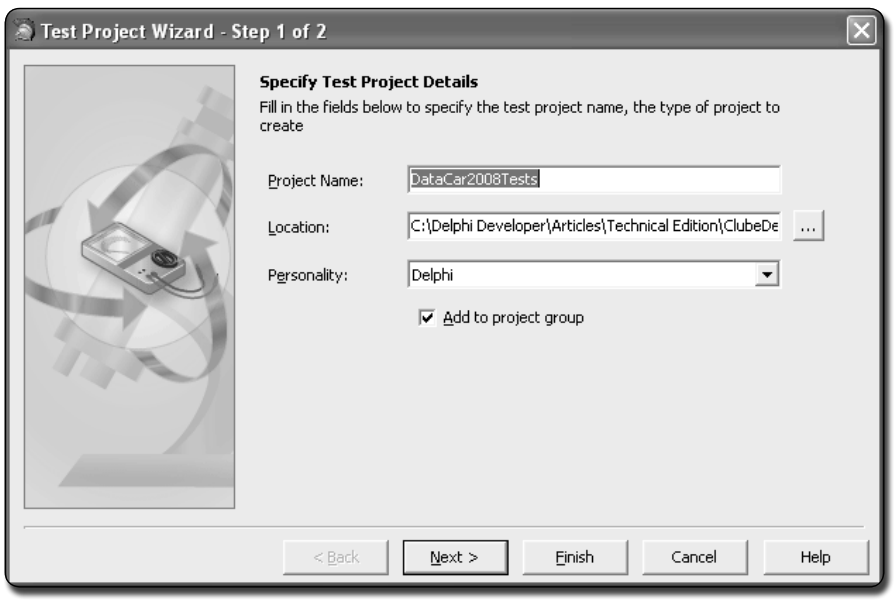


Figura 10. Assistente para criação de testes

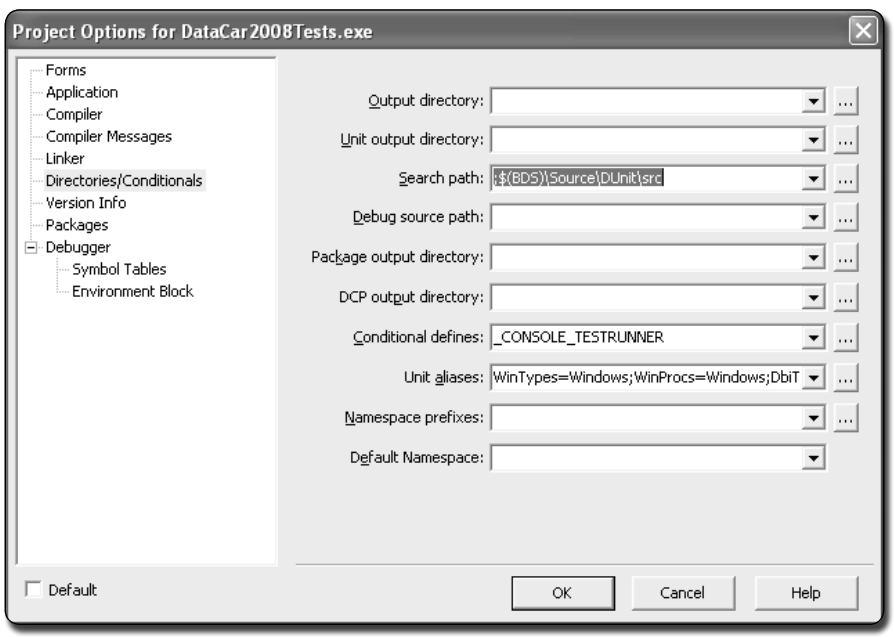


Figura 12. Localizando ModelU.pas

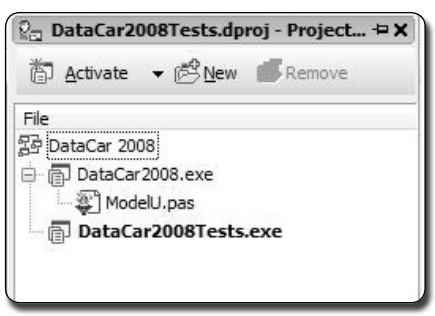


Figura 11. Project manager contendo 2 projetos

tudo isso, sua classe deve se parecer com a da **Listagem 2**.

Criamos um campo *FClasseBase*, que representa o objeto que vamos testar. O método *testValorDefaultId* irá utilizar *FClasseBase* para testar se o valor inicial do *ID* é mesmo zero. Como é bem sabido para utilizar um objeto temos que instanciá-lo, isso quer dizer que dentro do método *testValorDefaultId* teremos que instanciar *FClasseBase*? Não necessariamente. Veja que também sobrecarregamos os métodos *SetUp* e *TearDown*, eles são utilizados pelo *Framework* para configurar um ambiente antes e depois de um teste ser executado. Com isso podemos dizer que *SetUp* é disparado antes do teste rodar e *TearDown* depois. Nesses dois métodos vamos instanciar e destruir *FClasseBase*, como mostra a listagem **Listagem 3.**

Com isso asseguramos a disponibilidade do objeto *FClasseBase* para nosso teste, que é implementado na **Listagem 4.**

Realizando verificações

Os testes são compostos por *Checks* que o *Framework* disponibiliza. Na **Listagem 4** utilizamos um *Check* simples que verifica se a condição passada como parâmetro é verdadeira. Se for, o teste deu certo. Caso contrário a mensagem informada como segundo parâmetro é exibida.

Existem vários tipos de verificações disponíveis. Dentre elas temos algumas:

- procedure CheckTrue(condition: Boolean; msg: string = ''); virtual;
- procedure CheckFalse(condition: Boolean; msg: string = ''); virtual;
- procedure CheckEquals(expected, actual: extended; msg: string = ''); overload; virtual;
- procedure CheckEquals(expected, actual: extended; delta: extended; msg: string = ''); overload; virtual;
- procedure CheckEquals(expected, actual: integer; msg: string = ''); overload; virtual;
- procedure CheckEquals(expected, actual: string; msg: string = ''); overload; virtual;
- procedure CheckEqualsString(expected, actual: string; msg: string = ''); virtual;

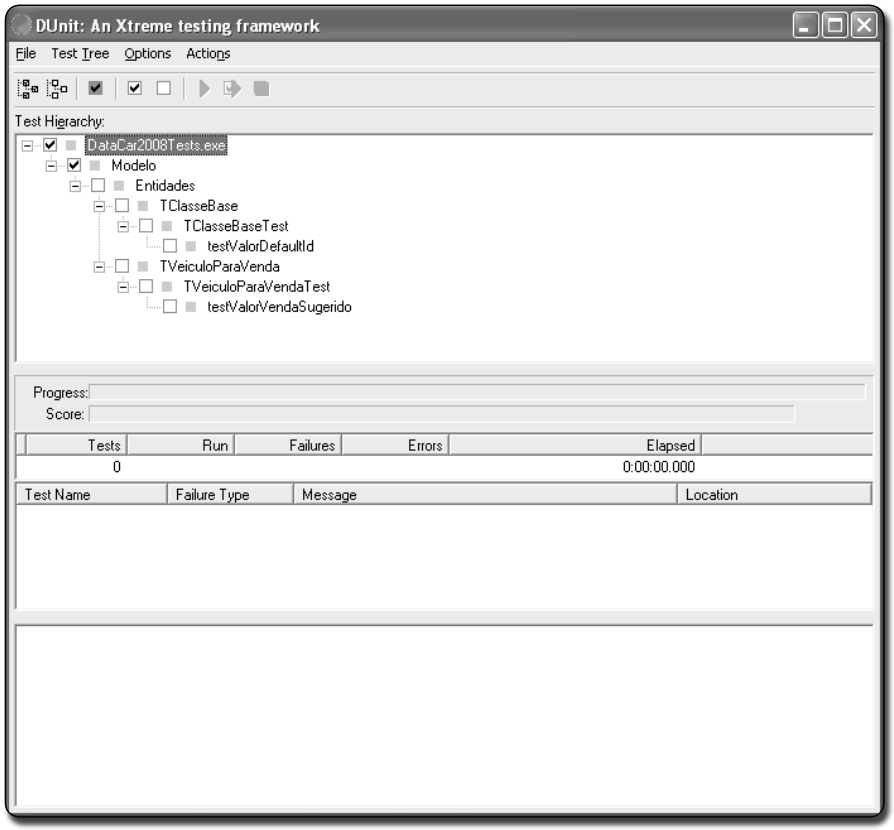


Figura 13. Dunit em ação

Listagem 2. Classe de teste

```
type
  TClasseBaseTest = class(TTestCase)
  protected
    FClasseBase: TClasseBase;
    procedure SetUp; override;
    procedure TearDown; override;
  published
    procedure testValorDefaultId;
  end;
```

Listagem 3. SetUp e TearDown

```
procedure TClasseBaseTest.SetUp;
begin
  inherited;
  FClasseBase := TClasseBase.Create
end;

procedure TClasseBaseTest.TearDown;
begin
  inherited;
  FreeAndNil(FClasseBase);
end;
```

Listagem 4. Teste implementado

```
procedure TClasseBaseTest.testValorDefaultId;
egin
  Check(FClasseBase.Id = 0,'Id deveria ser 0');
end;
```

Listagem 5. Registrando o teste

```
initialization
  RegisterTest('Modelo/TClasseBase', TClasseBaseTest.Suite);
```

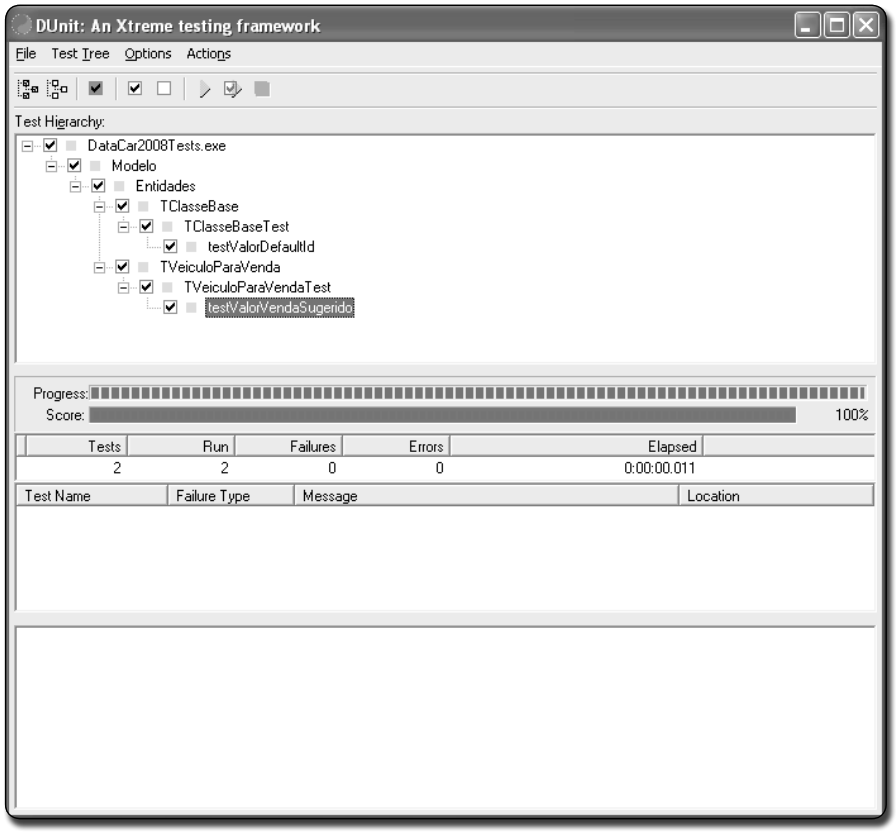


Figura 14. Testes executados

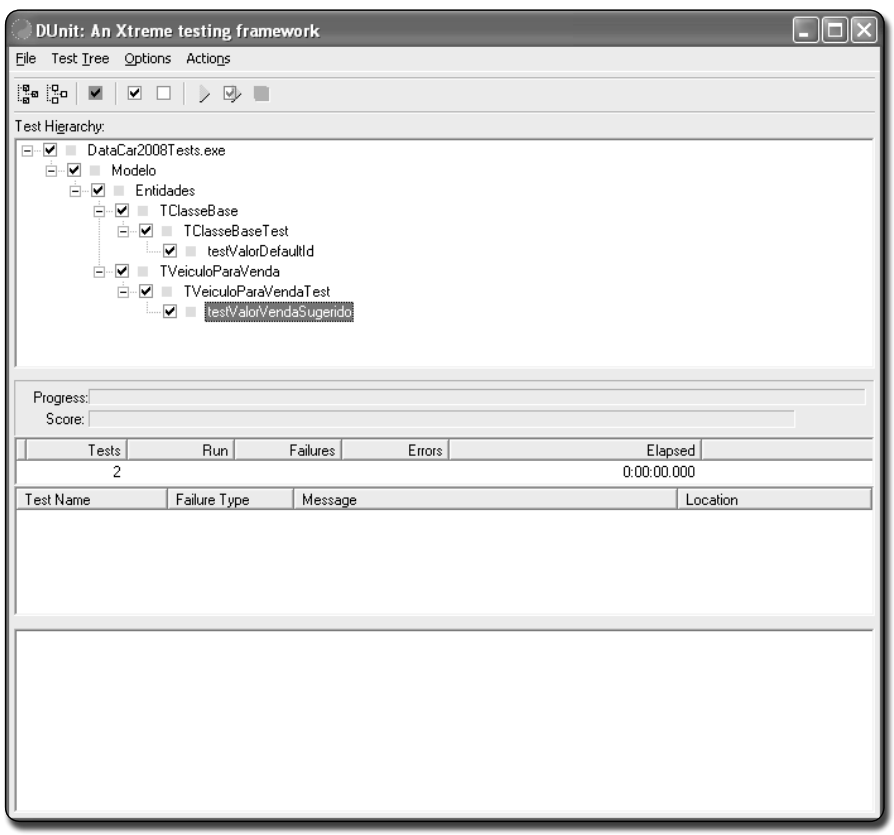


Figura 15. Os dois testes existentes sendo executados

A relação completa está na *Unit Test Framework* e cada um é utilizado de acordo com sua necessidade. Após termos escrito nosso primeiro teste é preciso registrá-lo para que o *Framework* possa executá-lo. Para isso na seção *Initialization* adicionamos o código da **Listagem 5.**

Executando o teste

Apenas execute o projeto *DataCar-2008Tests* e o gerenciado do *DUnit* será exibido, veja **Figura 13.**

Para executar agora é tudo muito intuitivo. Os testes selecionados são executados ao pressionar o botão *Play*. Veja o resultado na **Figura 14.**

O nosso teste *testValorDefaultId* passou, realmente o valor inicial do campo *ID* é *0*. Observação: caso o Delphi ainda acuse que arquivo *ModelU.dcu* não pode ser encontrado, copie ele para a pasta *Tests*.

Criando outro teste

Se voltarmos na especificação do nosso sistema veremos que temos um regra importante referente à venda de um veículo. Um veículo até a sua venda pode gerar despesas, e essas despesas junto com o lucro esperado do vendedor compõem o valor de venda sugerido. Vamos testar nossa classe *TVeículoParaVenda* refazendo os mesmos passos utilizados para criação do teste da *TClasseBase*. Na **Figura 15** temos o resultado do teste e na **Listagem 6** temos a Unit de teste.

Conclusão

Nesta primeira parte entendemos o problema, as regras de negócio que devemos aplicar no sistema e tornamos isso claro para nós e para o cliente utilizando a UML. De posse dos casos de uso criamos um diagrama de classes que representa nosso sistema e aplicamos testes unitários para garantir a qualidade do mesmo.

Quero que você leitor observe que ainda em nenhum momento falamos em banco de dados. Nosso foco esteve todo voltado em entender como o negócio funciona e como representá-lo e satisfazer as necessidades do cliente através de um sistema e esse

Listagem 6. TestCase de TVeiculoParaVenda

```
unit VeiculoParaVendaTest;
interface
uses TestFramework, ModelU, SysUtils;
type
  TVeiculoParaVendaTest = class(TTestCase)
  protected
    FVeiculo: TVeiculoParaVenda;
    procedure SetUp; override;
    procedure TearDown; override;
  published
    procedure testValorVendaSugerido;
  end;
implementation
{ TVendaTest }
procedure TVeiculoParaVendaTest.SetUp;
begin
  inherited;
  FVeiculo := TVeiculoParaVenda.Create;
end;
procedure TVeiculoParaVendaTest.TearDown;
begin
  inherited;
  FreeAndNil(FVeiculo);
end;
procedure TVeiculoParaVendaTest.
  testValorVendaSugerido;
var
 lDespesa1, lDespesa2: TDespesa;
begin
  FVeiculo.ValorEsperadoVenda := 10000;
  FVeiculo.LucroEstimado := 1000;
  lDespesa1 := TDespesa.Create;
  lDespesa1.Descricao := 'despesa 1';
  lDespesa1.Valor := 50;
  lDespesa2 := TDespesa.Create;
  lDespesa2.Descricao := 'despesa 2';
  lDespesa2.Valor := 100;
  FVeiculo.AddDespesa(lDespesa1);
  FVeiculo.AddDespesa(lDespesa2);
  check(FVeiculo.ValorSugeridoVenda = 11150, 'O valor deveria ser R$ 11.150,00');
end;
initialization
  RegisterTest('Modelo/TVeiculoParaVenda', TVeiculoParaVendaTest.Suite);
end.
```

deve ser o principal objetivo nosso. Como, onde ou de que forma esses dados serão armazenados não deve ser nossa primeira preocupação. Aplicar a orientação a objetos aliada a UML e Testes unitários nos conduz a isso. Caso nosso cliente já desejasse ver um algo em funcionamento, já temos as regras funcionando, bastando apenas criar um protótipo simples, que guarda tudo em memória.

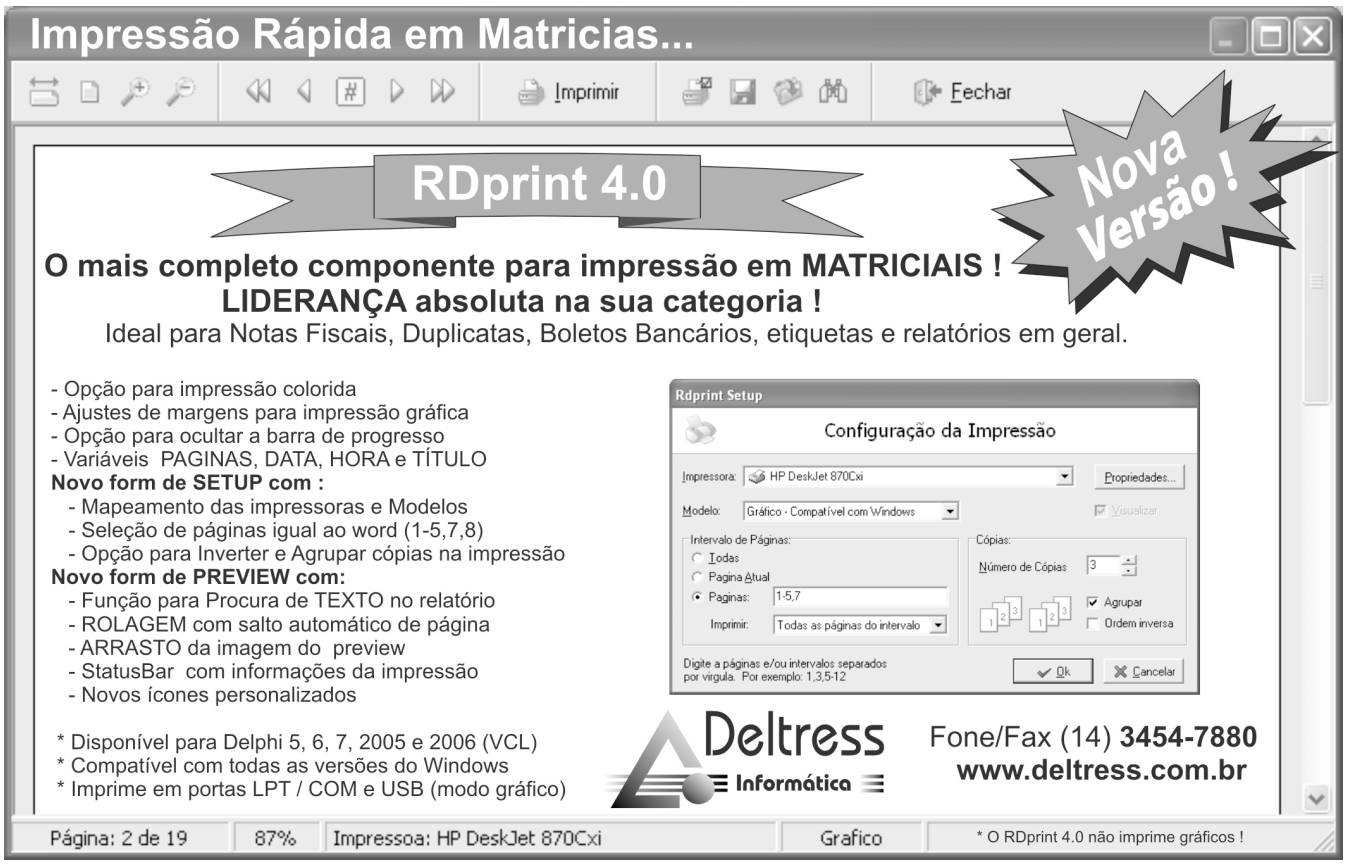
Na próxima parte do artigo vamos ver como salvar fisicamente nossos objetos, utilizando para isso o padrão DAO. Fique ligado e até lá. ●

Dê seu feedback sobre esta edição!

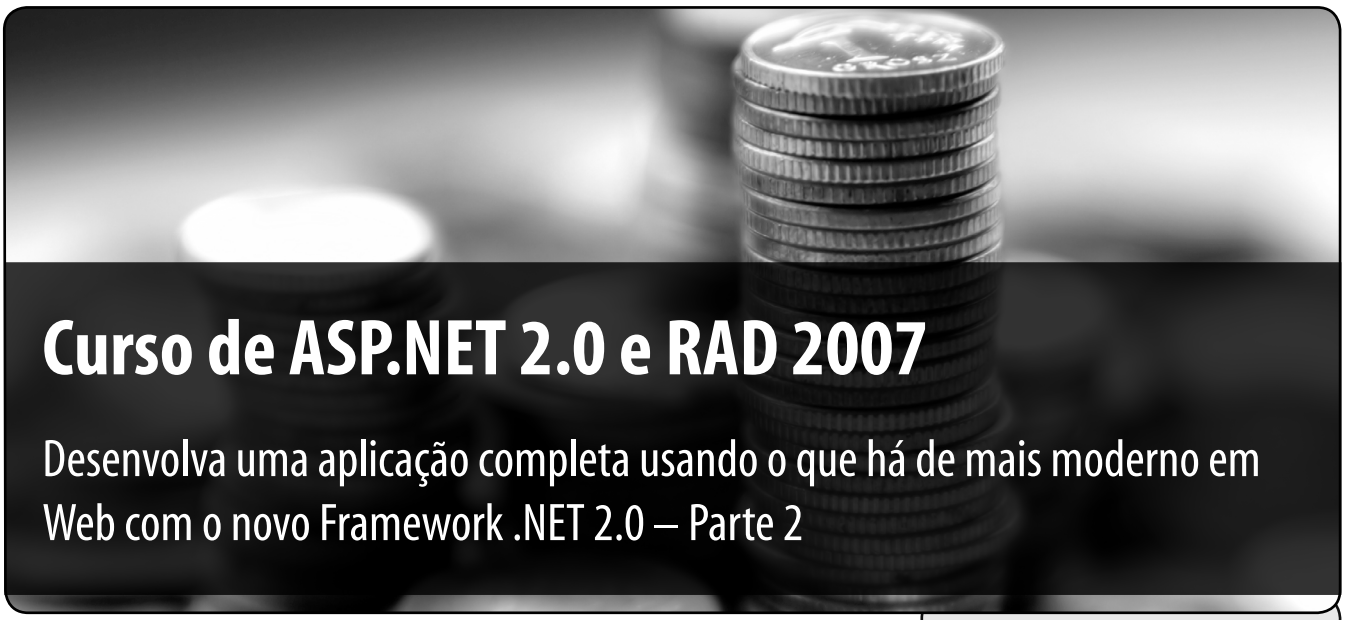
A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET



Curso de ASP.NET 2.0 e RAD 2007

Desenvolva uma aplicação completa usando o que há de mais moderno em Web com o novo Framework .NET 2.0 – Parte 2

Nesse artigo veremos

- Novidades do ASP.NET 2.0;
- Segurança;
- Solicitação de orçamento on-line.

Qual a finalidade

- Criação da parte de segurança do Web Site impedindo, por exemplo, que usuários não administradores acessem páginas de cadastros como produtos ou categorias.

Quais situações utilizam esses recursos?

- A segurança do site é parte primordial nos dias de hoje. O que veremos nesse artigo faz referência à manutenção dos dados de nosso Web Site.

Adriano Santos
falecom@adrianosantos.pro.br
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É editor técnico, colunista e membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphi-todelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Guinther Pauli
guinther@devmedia.com.br
é Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine e ClubeDelphi.

Nota

Na primeira parte desse mini-curso, criamos as telas de cadastro de categorias e produtos. Essas telas não precisam estar acessíveis a qualquer usuário, já que não há necessidade de internautas comuns cadastrarem, alterarem ou mesmo excluírem esses itens. E é sobre esse tema que veremos e discutiremos a seguir. Através do ASP.NET Configuration Site, faremos a inclusão de usuários, roles (grupos de usuários e regras) e definiremos como nosso Web Site se comportará mediante as credenciais impostas pela aplicação.
Faremos também o desenvolvimento das telas de login para que o internauta possa cadastrar-se e/ou logar-se em nosso site.

Na primeira parte do curso de ASP.NET 2.0 com RAD Studio 2007, vimos os principais recursos que a nova versão do IDE nos proporciona. Com apenas alguns cliques e praticamente sem a adição de nenhum código, conseguimos criar as telas de cadastro e manutenção para as tabelas de produtos e categorias. Passamos também pela criação das *Master Pages*, páginas mestres que substituem os antigos controles *User Controls* utilizados até o .NET 1.1 e presentes no Borland Developer Studio 2006.

Em continuação ao Mini-Curso de ASP.NET, apresentaremos a seção de segurança e solicitação de orçamento. É importante ficar atento ao processo passo a passo, para que você possa executar o Mini-Curso e aproveitar ao máximo o que será apresentado.

Nesse artigo veremos como utilizar os controles *LoginView*, *LoginStatus*, *CreateUserWizard* e *LoginName*. Todos esses controles fazem parte da paleta de componentes *Web Login*. Com eles podemos facilmente validar a entrada de usuários em nosso *Web Site*. Além disso é possível trabalharmos com *Roles*, ou seja, grupos de usuários e permissões. Outro item bastante importante é a configuração dessas permissões no ASP.NET *Configuration*. Nele podemos gerenciar os usuários, grupos e negar ou permitir o acesso a determinadas páginas do site.

Configurando a segurança

Através do *ASP.NET Configuration Site*, podemos trabalhar toda a segurança da aplicação. Abra o projeto criado no artigo anterior e então clique com o botão direito do mouse sobre o projeto no *Project Manager*. Utilize a opção *ASP.NET Configuration* (**Figura 1**).

Clique na guia *Security*, onde a primeira configuração necessária é a autenticação, que podemos trabalhar de duas formas:

• *Internet (Forms)*: a autenticação é baseada em um formulário de *login* e a autenticação feita "na mão" pelo desenvolvedor;

• *Intranet (Windows)*: a autenticação é baseada no *login* na rede, ou seja, não necessita de nenhum trabalho de autenticação. É mais utilizado em aplicações corporativas.

Vale lembrar que muitas configurações de segurança usarão um *BD* especial criado automaticamente pelo ASP.NET, o *aspnetdb*.

Clique em *Select Authentication Type* localizado em *Home*. Altere a configuração para *From the Internet*, pois trabalharemos com o tipo *Forms*, e clique em *Done*. Com isso já temos toda a parte de autenticação pronta, isso mesmo, pronta!

Volte ao RAD Studio 2007 (sem fechar o site de configuração) e entre na *MiniCurso.master*. Abaixo do *SiteMap*, adicione um *LoginView*, localizado na categoria *Web Login* da paleta de componentes, e arraste-o para a página. Esse controle tem como objetivo criar *Multiviews* ("multi visualizações") de acordo com o *status* do usuário (*Logado* ou *Anônimo*).

A visualização *default* é a anônima, então arraste um controle *Login* sobre o *LoginView*, formate conforme seu gosto e altere a propriedade *Orientation* para *Horizontal*. Selecione a *Smart Tasks* do *LoginView* e altere o *View* para *LoggedInTemplate*. Digite um texto, por exemplo: "Olá" e arraste ao lado o controle *LoginName*.

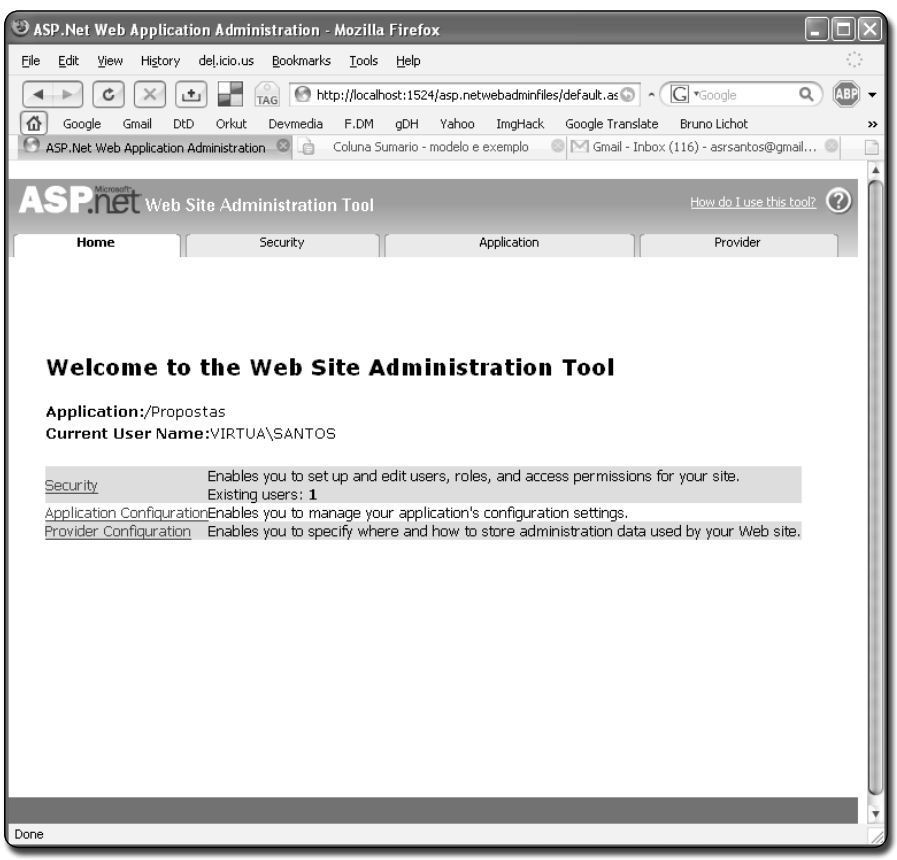


Figura 1. Página ASP.NET Configuration

Nota do DevMan

Por padrão os componentes da paleta Web Login possuem seus Label's em inglês. Isso pode ser facilmente modificado, pois há diversas propriedades que podemos modificar e traduzir para a linguagem local. Vejamos o que podemos modificar para que o usuário não se perca na hora de efetuar o login ou mesmo cadastrar-se. Pressione F11 com o componente LoginView selecionado e veja suas propriedades:

• UserNameLabelText: Troque de User Name: para "Nome de Usuário:";
• PasswordLabelText Modifique para "Senha:";
• PasswordRequiredErrorMessage: "Senha obrigatória";
• UserRequiredErrorMessage: "Nome de Usuário obrigatório";
• RememberMeText: "Lembrar meu usuário";

Note que, ao modificar a propriedade, o template é automaticamente atualizado na IDE do RAD Studio 2007. Além disso, perceba que outras propriedades podem sofrer modificações, tais como: FailureText, texto apresentado quando alguma falha de login ocorre e LoginButtonText, ou seja, o texto do botão Login.

A seguir veremos a inclusão do controle CreateUserWizard que também pode ser totalmente modificado para a linguagem local. Você contará com algumas propriedades importantes, como por exemplo:

• UserNameLabelText: Texto para o Label do TextBox de nome de usuário;
• UserNameRequiredErrorMessage: Mensagem apresentada ao requerer nome de usuário;
• UnknownErrorMessage: Mensagem de erro desconhecido;
• QuestionLabelText: Label para o campo de pergunta secreta;
• QuestionRequiredErrorMesasge: Mensagem apresentada ao requerer nome de usuário;
• PasswordLabelText: Texto para o Label do TextBox de senha;
• PassowordRequiredErrorMessage: Mensagem de texto ao requerer senha;

E ainda os textos dos botões:

• ContinueButtonText: Texto do botão Continue ("Continuar");
• CreateButtonText: Texto do botão Create User ("Confirmar criação de usuário");
• FinishButtonText: Texto do botão Finish ("Finalizar");
• StartNextButtonText: Texto do botão Next ("Próximo passo");

Entre outras mensagens e textos que podem ser modificados. Em resumo, os controles do ASP.NET no RAD Studio 2007 podem ser personalizados em praticamente tudo, facilitando a vida do desenvolvedor e agilizando o processo de criação e manutenção do site.

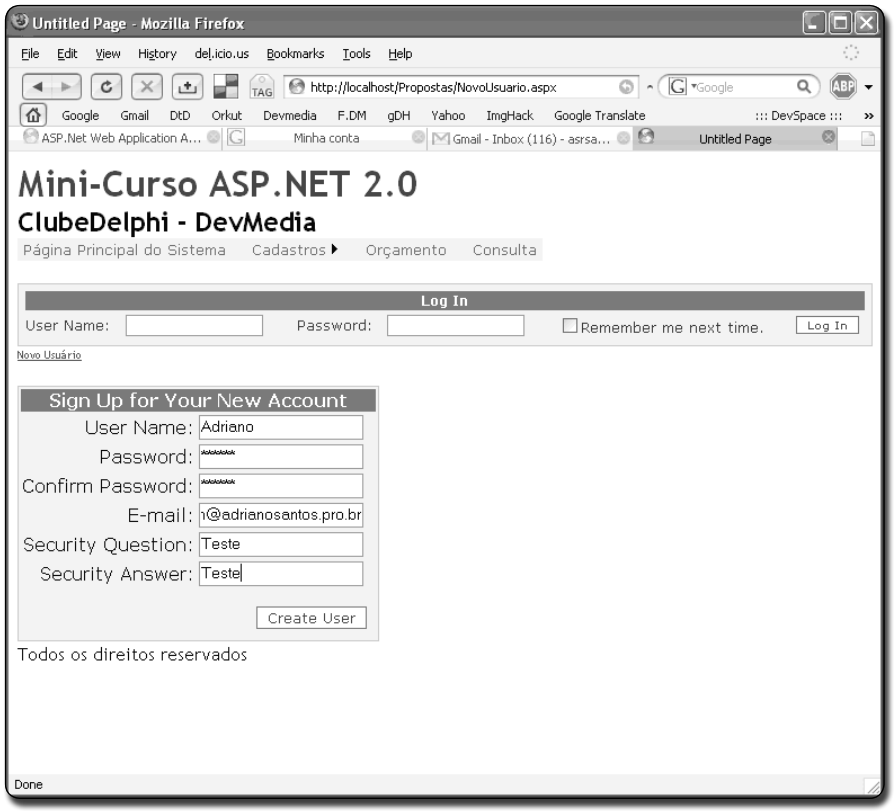


Figura 2. Cadastro do Usuário

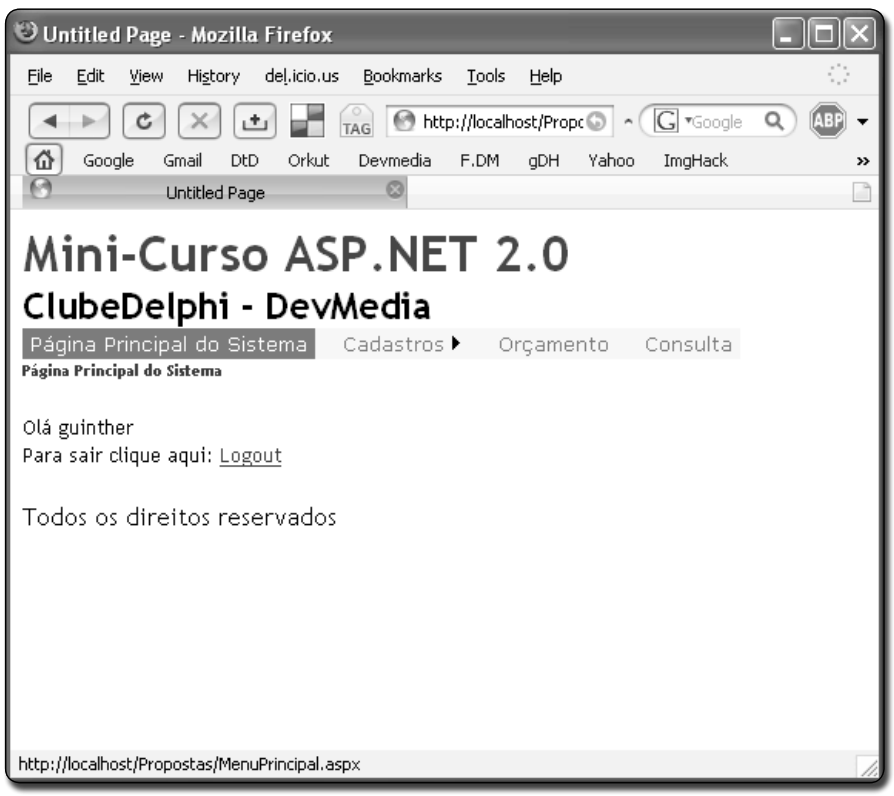


Figura 3. Usuário Logado

Logo abaixo adicione o texto "para sair clique aqui: " e arraste ao lado o controle *LoginStatus*. Pronto, temos a tela de *login* e recepção do usuário. Só falta então ter o usuário com um cadastro de usuários.

Adicionando usuários

Adicione ao projeto uma nova página chamada "NovoUsuario.aspx" usando o menu *File>New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Content Page*. No *Content* da página arraste o controle *CreateUserWizard*, ainda na categoria *Web Login*.

A única propriedade que vamos mudar é *ContinueDestinationPageUrl* para "~/MenuPrincipal.aspx", pois com isso ao finalizar o cadastro, o usuário é redirecionado à página principal. Novamente na *Master Page*, adicione no *LoginView* de visualização anônima (altere a *view* do controle para *AnonymousTemplate*) um *Hyperlink* localizado na paleta *Web-Controls* e altere a propriedade *Text* para "Novo Usuário" e *NavigateURL* para "~/NovoUsuario.aspx".

Finalmente podemos simular algo. No *Project Manager*, clique com o botão direito do mouse sobre *MenuPrincipal.aspx* e *View in Browser*. Efetue o procedimento de cadastro clicando no link *Novo Usuário* (**Figura 2**).

Clique em *Create User*, então o cadastro é finalizado. Clique em *Continue*, com isso é feito o redirecionamento ao *Menu Principal* e temos o resultado conforme a **Figura 3**.

> **Nota:** Por padrão, o ASP.NET usa uma política de segurança para as senhas, que devem ser maiores que sete dígitos e requerem um caractere especial (*, &, % etc.).

Efetue também testes através do *login* do site. Existem outros *templates* prontos que podemos utilizar, como:

- *PasswordRecovery*: lembrar a senha do usuário através de envio de e-mail;
- *ChangePassword*: alterar a senha do usuário.

Trabalhando com grupos de usuários

Podemos trabalhar também com grupos de usuários, facilitando o gerenciamento de permissões no site. Volte ao site de configuração e clique em *Enable roles*. Com isso podemos criar grupos clicando em *Create or Manage roles*. Crie um grupo chamado "Administrativo" (digite o nome e clique em *Add Role*) e clique em *Back*. O objetivo é criar uma regra onde somente usuários desse grupo possam cadastrar categorias e produtos.

Novamente no site de configuração, clique em *Create access rules*. Selecione o diretório *Cadastro*, marque em *Role* a opção *Administrativo* e em *Permission*, marque *Allow* (**Figura 4**).

Assim estamos dando direito de acesso aos membros do grupo *Administrativo*. Finalize, clicando em *Ok*. Clique agora em *Manage Access Rules*, sendo exibida configuração atual. Estamos permitindo todos os usuários além dos usuários do grupo *Administrativo*. Clique em *Add new access rules*, selecione *All Users* e *Deny* e clique em *Ok* (certifique que o diretório selecionado é o *Cadastro*). Agora sim, a pasta cadastros está segura.

Selecione o diretório *Orcamento*, clique em *Add new access rules*, marque a opção *Anonymous users*, *Deny* e clique em *Ok*. Com isso somente usuários *logados* entram nessa pasta. Simule na aplicação a tentativa de acesso às páginas, sabendo do erro de falta de permissão nas páginas.

Para adicionar um usuário a um grupo, na tela principal de *Security* no site de configuração, acesse o *Manage Users*. Na listagem de usuários, clique em *Edit Roles* e selecione o grupo desejado de cada usuário.

Quando entramos em um menu sem permissão, é redirecionado para uma página *login.aspx*, essa é a configuração *default*. Para alterar, abra o arquivo *web.config* e altere a tag *<authentication mode="Forms" />* conforme o seguinte código:

```
<authentication mode="Forms">
  <forms loginUrl="MenuPrincipal.aspx"/>
</authentication>
```

Finalmente temos a segurança completa no nosso site, muito mais fácil do que antes do ASP.NET 2.0.

Orçamento

Como já mencionado nas partes anteriores, o usuário do sistema poderá gerar orçamentos dos produtos que fornece para o cliente. Para isso, selecionará os dados do orçamento e os itens que foram solicitados pelo cliente.

Clique com o botão direito na pasta *Orcamento*, selecione a opção *Add New>Other>Delphi .NET Projects>New ASP.NET Files>ASP.NET Content Page*. Na tela seguinte informe "Orcamento.aspx" no campo *Name*, selecione a opção *Master page file* e clique *Ok*.

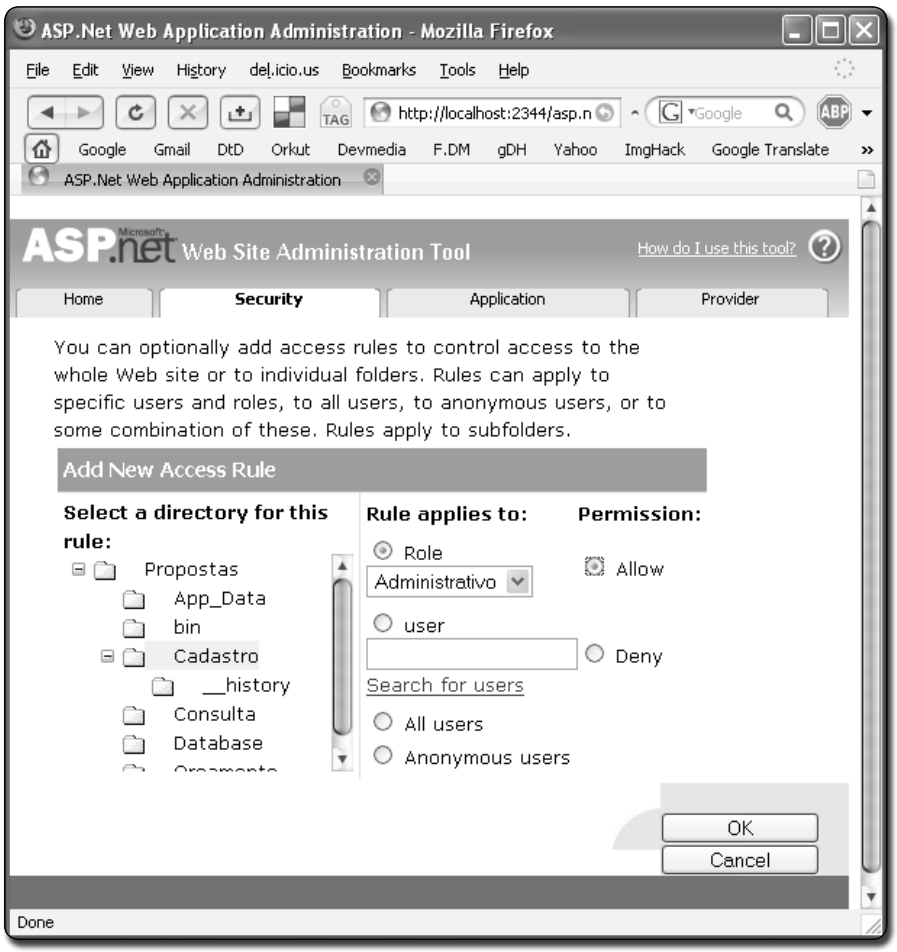


Figura 4. Configurando os acessos

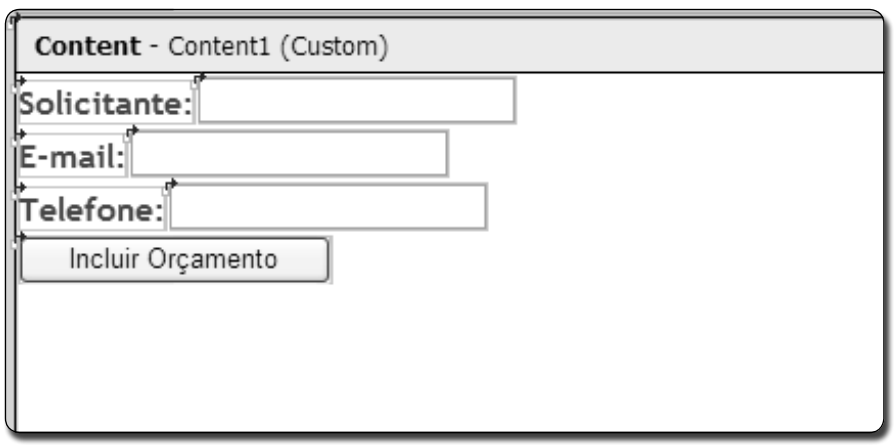


Figura 5. Incluindo os campos para registro do Orçamento

Após a seleção da *Master Page*, a página é aberta para iniciar a construção do orçamento. Essa tela terá um conjunto de controles, que informarão os dados do Orçamento, e um *GridView* contendo os produtos disponíveis para venda, podendo selecionar os itens em um *CheckBox*.

Selecione então, da paleta *WebControls*, três *TextBoxes* nomeando-os, pela propriedade *ID*, para "txtSolicitante", "txtEmail" e "txtTelefone", respectivamente. Inclua também três *Labels e* um *Button*("btnIncluir"). Formate-os semelhante à **Figura 5**.

Além dos controles já adicionados, ainda será necessário selecionar os produtos que farão parte do orçamento. Para isso, o usuário fará a seleção a partir de um *GridView*, utilizando um *CheckBox* presente em uma das colunas. Arraste então um *GridView*, crie uma conexão com o banco de dados, semelhante ao que já fizemos nos outros exemplos, mas escolha a opção *Specify a custom SQL statement or stored procedure* (**Figura 6**).

Nessa tela você poderá informar uma instrução SQL personalizada ou até mesmo informar uma *Stored Procedure* existente. Nesse exemplo, utilizaremos o código da **Listagem 1** (na aba *Select*).

A instrução SQL da **Listagem 1** faz a seleção dos produtos e suas respectivas categorias para exibição no *GridView*. Com isso, teremos a listagem dos produtos exibindo a descrição da categoria de cada produto. As outras guias poderiam ser utilizadas para cada operação de sua fonte de dados, colocando em cada Guia (*Select, Insert, Update* e *Delete)* o SQL correspondente.

Listagem 1. Código para retornar o produto, valor e sua descrição

```
SELECT
  PRODUTOS.PRODUTOSID,
  CATEGORIA.DESCRICAO AS CATEGORIA,
  PRODUTOS.DESCRICAO AS DESCRICAO,
  PRODUTOS.VALOR
FROM
  PRODUTOS
  INNER JOIN CATEGORIA ON
  PRODUTOS.CATEGORIAID = CATEGORIA.CATEGORIAID
```

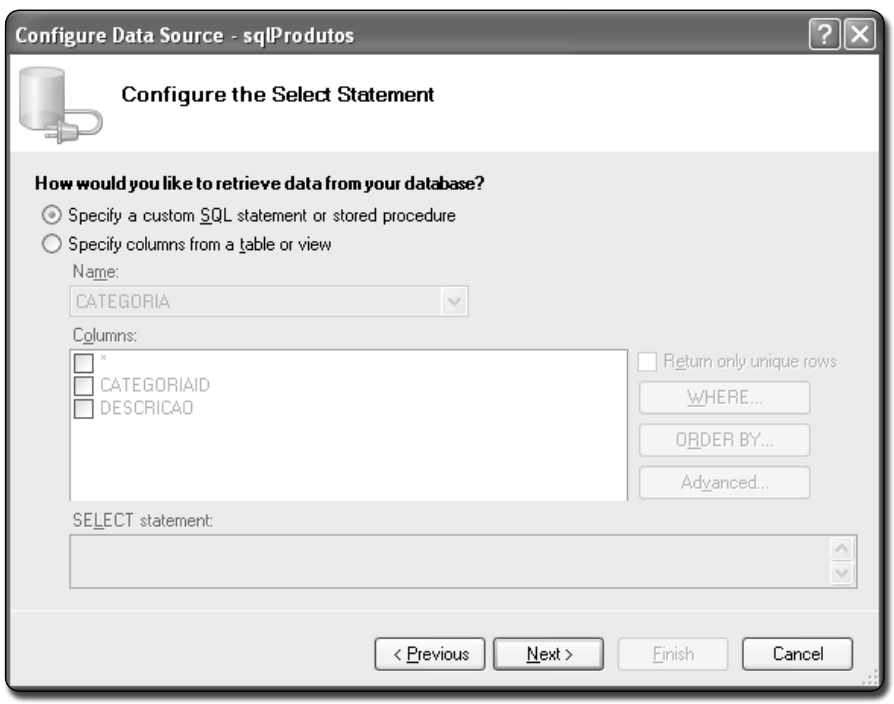


Figura 6. Selecionando uma instrução SQL personalizada

Além das colunas que serão criadas automaticamente pelo controle, precisaremos incluir a coluna do *CheckBox* para seleção dos produtos a fim de concluir o orçamento. Para isso, precisamos adicionar uma coluna do tipo *TemplateField*. Clique na *Smart Tasks* do *GridView* e escolha a opção *Add New Column*.

Selecione *TemplateField* em *Choose a field type* informe "Selecione" em *Header Text* e clique *Ok*. Agora que a coluna já está criada, é necessário adicionar o *CheckBox*. Para isso, clique novamente na *Smart Tasks* e selecione a opção *Edit Template*.

Essa funcionalidade mostra as diversas seções de uma coluna *TemplateField*. Por padrão, a seção *ItemTemplate* é apresentada e é exatamente ela que é exibida na carga do *GridView*. Assim, arraste um *CheckBox* para dentro dessa seção, configure a propriedade *ID* como "chkSelecione" para que depois consigamos recuperar os valores selecionados. Veja na **Figura 7**, como deve ficar o *Template*. Após esse passo, clique novamente na *Smart Tasks* e escolha a opção *End Template Editing*.

Já temos quase tudo pronto, só falta codificar para que sejam recuperados os *CheckBoxes* selecionados. Faremos isso no evento *Click* do botão *Incluir Orçamento*, que também enviará um e-mail para o administrador do sistema para que avalie o orçamento e retorne para o cliente.

A classe que envia e-mails no .NET Framework 2.0 é a *System.Net.Mail* e trabalharemos com a mesma para enviar e-mail através da aplicação. Para isso, clique duas vezes no botão *Incluir Orçamento* e codifique conforme mostra

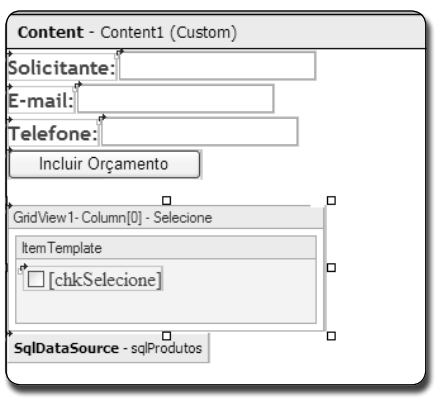


Figura 7. Configurando o CheckBox para seleção dos produtos

Listagem 2. Verificando os produtos selecionados e enviando e-mail

```
procedure TOrcamento.btnIncluir_Click(sender:
  System.Object; e: System.EventArgs);
var
  idProduto: Integer;
  Body: string;
  chk : CheckBox;
  I : Integer;
  item : GridViewRow;
  mailFrom : MailAddress;
  mailTo : MailAddress;
  email : MailMessage;
  smtp : SmtpClient;
begin
  body := 'Produtos Selecinados: \n';
  for I := 0 to GridView1.Rows.Count - 1 do
  begin
    item := GridView1.SelectedRow;
    chk := CheckBox(item.Cells[0].
      FindControl('chkSelecione'));
    if chk.Checked then
    begin
      idProduto := Convert.ToInt32(item.
        Cells[1].Text);
      body := body + 'Código: ' +
        idProduto.ToString() + '\n';
      body := body + 'Descrição do Produto: ' +
      item.Cells[3].Text + '\n';
    body := 'Valor do Produto: R$ ' +
      Convert.ToDouble(item.Cells[4].Text)
      .ToString + '\n\n\n';
    end;
  end;

  mailFrom := MailAddress.Create(
    'minicurso@minicurso.com.br',
    'Mini Curso ASP.NET');
  mailTo := MailAddress.Create(
    txtEmail.Text, txtSolicitante.Text);
  email := MailMessage.Create(mailFrom, mailTo);
  email.Subject := 'Orçamento OnLine';
  email.Body := body;
  smtp := SmtpClient.Create;
  smtp.Host := 'smtp.suaempresa.com.br';
  smtp.Port := 25;
  smtp.Send(email);
end;
```

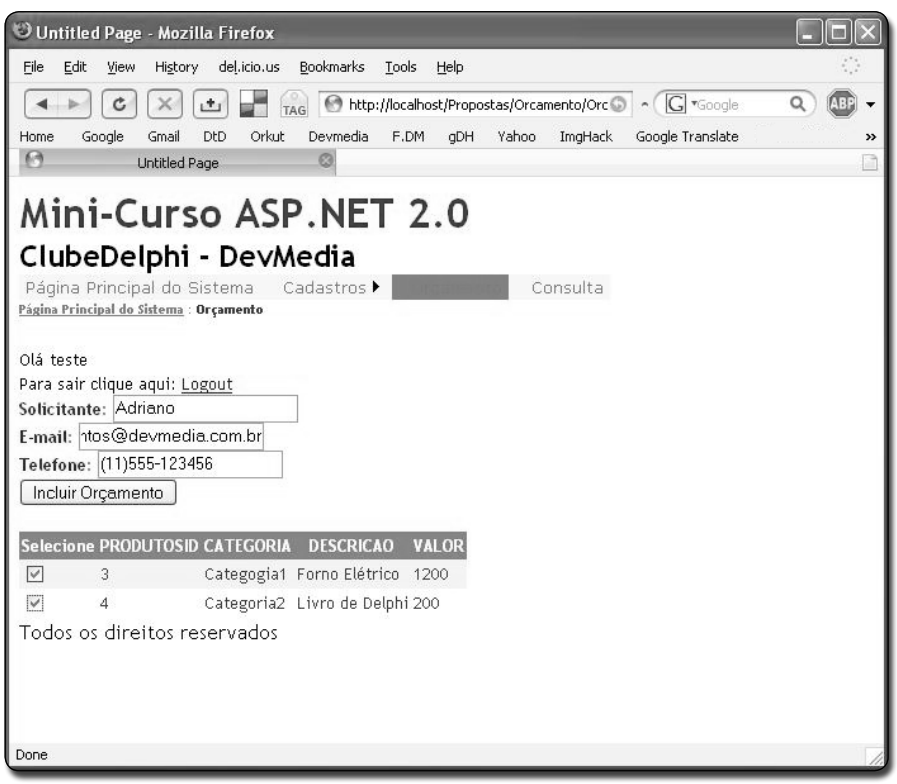


Figura 8. Aplicação em execução, selecionando os produtos no GridView

a **Listagem 2.** Inclua o *namespace* mencionado na seção *Uses* do *WebForm*.

Após ter codificado, vamos entender o que o código faz. Declaramos algumas variáveis para nos auxiliar em todo o processo de montagem e envio do e-mail. A variável *body* é responsável por receber o corpo do e-mail. Nela, estamos incluindo os produtos e respectivos valores selecionados no *GridView*. Em seguida montamos um laço *for* responsável por percorrer o *GridView* e verificar se o *CheckBox* associado ao produto encontra-se selecionado. Em caso positivo alteramos o *body* incluindo mais um produto.

Após a finalização do *for*, atribuímos às variáveis *mailFrom* e *mailTo* a criação dos objetos *MailAddress* informando, em seus parâmetros, o remetente ("mailFrom") e o destinatário ("mailTo"). Criamos também uma variável *email* passando para ela *remetente* e *destinatário*. Por fim, o objeto *smtp* é criado e configurado para envio. O método *Send* é usado para, efetivamente, enviar a mensagem (**Figura 8**).

Conclusão

Como você pôde perceber, o desenvolvimento com ASP.NET 2.0 no RAD Studio 2007 está bem simples. Entenda os conceitos, aplique as técnicas corretamente e terá uma aplicação eficaz e de simples manutenção. Além disso, há diversos novos controles que fazem a diferença na hora de desenvolver. A agilidade e facilidade de manutenção do código-fonte da aplicação faz com que o ASP.NET 2.0 seja uma excelente solução para quem necessitar desenvolver Web Sites com rapidez. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

RSS no RAD Studio 2007

Entendendo e implementando RSS em suas aplicações

Nesse artigo veremos

- O que é RSS;
- Como criar um gerador de RSS;
- Como criar um leitor de RSS.

Qual a finalidade

- Como poderemos ver, as notícias em RSS são importantíssimas, pois permitem que o internauta fique por dentro das novidades sem precisar vasculhar uma série de sites.

Quais situações utilizam esses recursos?

- Rapidez e agilidade na notícia, essa é a principal razão da utilização de RSS. Esse padrão veio para ficar, pois com ele nos atualizamos rapidamente sem precisar ir atrás da notícia e/ou informação. Praticamente todos os portais de informação, entretenimento etc. possuem seus links RSS para leitura rápida.

Manoel Edesio B da Silva
manoel.edesio@hkconsultoria.com.br
detém as certificações em Borland C++ Builder, Borland Delphi for Win32 e Borland Delphi for .NET. Palestrou para a Borland em Delphi Meetings em alguns estados brasileiros, palestrante na Borland Conference de 2004, 2005, 2006 e 2007. Atualmente é diretor da HK Consultoria, empresa especializada em consultoria, treinamentos customizados e desenvolvimento de software em produtos Borland. Além desses quesitos a HK atua com projetos especializados em mobilidade com utilização de celulares e PDA'S tornando-se em 2008 parceira PDA da operadora Claro. Maiores informações sobre a empresa em www.hkconsultoria.com.br.

Nesse artigo aprenderemos as principais características do RSS, o que é exatamente esse recurso e ainda veremos como funciona a estrutura básica de um arquivo .rss, também chamado de FEED.

Para exemplificar a utilização de RSS, faremos duas aplicações em Delphi 2007, onde criamos uma aplicação para geração de RSS e outra para a leitura desses arquivos.

O que é RSS?

O RSS ("Rich Site Summary ou Really Simple Syndication") é um padrão que surgiu no mercado com o intuito de fazer com que as notícias corram atrás de você e não você atrás delas. Isso mesmo, vamos a uma explicação mais detalhada sobre o assunto.

Esse recurso surgiu para facilitar a publicação de informações na internet. Criado pela Netscape, o RSS é visto como uma forma ideal para se compartilhar conteúdos e além de tudo manter-se a par das novidades. O recurso é composto por um arquivo XML ("eXtensible Markup Language") que segue um padrão de acordo com suas respectivas versões. Através desse recurso podemos disponibili-

zar informações em nosso site onde o conteúdo estará em um arquivo XML de extensão .rss (o arquivo também poderá ter extensões diferentes, essa é a mais comum). Quando o usuário desejar ler o conteúdo do nosso arquivo RSS ele irá utilizar um leitor RSS e informará nele o link da mesma. Essa aplicação tem o objetivo de ler o conteúdo do arquivo e exibir para o usuário em sua interface de maneira a facilitar o seu uso no dia-a-dia. Geralmente esses aplicativos marcam as notícias já lidas, informam sobre novas notícias etc.

Nesse artigo veremos os principais conceitos sobre o RSS e como criar e ler tais arquivos em nossas aplicações.

Entendendo a estrutura do arquivo

Para iniciarmos o nosso aprendizado, vamos primeiramente entender a estrutura de um arquivo RSS conforme a nossa **Listagem 1**. Primeiramente, no início do arquivo, temos o elemento *<rss>* definindo que se trata de um RSS e na seqüência temos o atributo que determina a versão do mesmo. Esse valor sempre depende da versão utilizada na construção do arquivo e é obrigatório. Em seguida temos uma *Tag* chamada *<channel>* que se trata da *Tag* principal, pois ela envolverá todos os demais itens ali contidos, lembrando sempre que ela só aparece uma única vez no arquivo. Seu propósito é descrever todo o conteúdo do arquivo RSS.

Dando seqüência, temos a *Tag <title>* descrevendo o título do documento, lembrando que a idéia aqui é descrever realmente informações sobre o documento em geral e não sobre uma notícia em específico. Em *<link>* informamos a URL do documento que identifica esse canal de notícias. Depois vamos encontrar o *<description>*, ou seja, a descrição breve das informações do arquivo.

Caso existam informações de *Copyright* ("Direitos autorais") eles são colocados em *<copyright>*. Prosseguimos então com a primeira etapa da estrutura do arquivo, o *<generator>*, onde podemos definir a aplicação que gerou o RSS ou o recurso utilizado para tal.

Em seguida temos as *Tags* correspondentes aos sub-itens do documento iniciando por *<item>* que vai representar sempre as notícias, as informações, ou seja, os itens de conteúdos disponibilizados. No caso do elemento item o número de ocorrências dentro do arquivo sempre irá variar de acordo com o número de notícias que desejamos informar. Dentro de *<item>* temos vários sub-itens que são opcionais, porém a *<title>* ou *<description>* obrigatoriamente deverão aparecer para a conclusão da formação do item. Em *<title>* informamos a URL de origem e em *<description>* a descrição detalhada dele. Por fim, encontramos o *<pubDate>*, onde informamos a data de publicação, sempre respeitando o padrão RFC822 "Quinta-Feira, 5 de Junho de 2008 08:05:00 GMT ".

Nota: Em se tratando de RSS podemos construir aplicações que farão a geração do arquivo ("Geradores de RSS") e também desenvolver os Leitores de RSS ("Feeders"). Em nosso artigo vamos tratar do desenvolvimento de leitores de RSS para nossos arquivos e também para leitura deles diretamente de links presentes na internet.

Exibindo um RSS

Como primeiro projeto vamos criar uma aplicação ASP.NET e exibir o conteúdo de um arquivo RSS que criaremos baseado em nossa **Listagem 1**. Então vamos lá, coloquem a mão na massa. No RAD Studio acesse o menu *File>New>ASP.NET Web Application – Delphi for .NET* e na seqüência salve nosso projeto como "RSSExibir.dproj" mantendo a pasta de localização do projeto como *c:\Inetpub\wwwroot\RSSExibir* e o servidor para testes *Cassini Web Server*.

Após configurar a aplicação clique no botão *Ok* e vamos acessar o *Project Manager* localizado no lado direito do IDE. Clique com o botão direito do mouse sobre o nosso projeto *RSSExibir*.

Listagem 1. Estrutura de um arquivo (.RSS)

```
<rss version="2.0">
<channel>
  <title>HK Consultoria - Novidades</title>
  <link>www.hkconsultoria.com.br</link>
  <description>Últimas Notícias</description>
  <copyright>Manoel Edesio</copyright>
  <generator>My Generator</generator>
  <item>
    <title>HK Consultoria é citada pela Borland em
      projeto WAP</title>
      <link>http://www.hkconsultoria.com.br/noticias.asp
      </link>
    <description>
      Integrada ao ERP da companhia, a solução será
      oferecida para a sua base de clientes, formada
      por empresas de segurança e limpeza, todas com
      atuação nacional. A HK Engenharia - empresa
      especializada em sistemas de gestão para
      prestadores de serviço de segurança e limpeza -
      escolheu o Delphi, da CodeGear - unidade da
      Borland voltada aos desenvolvedores - para a
      construção do Sistema de Alocação de Recursos
      Ponto Real (SARPR) para o controle remoto de
      cartões de ponto, que pode ser acessado via
      internet ou WAP. O seu lançamento está previsto
      para o início de 2008.
    </description>
    <pubDate>01/01/2008</pubDate>
  </item>
</channel>
</rss>
```

Listagem 2. Implementação do Page_Load do WebForm

```
procedure TDefault.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  FArquivoRSS: System.IO.StreamReader;
begin
  FArquivoRSS := StreamReader.Create(Server.
    MapPath('MyRSS\HKRSS.rss'));
  Response.ContentType := 'text/xml';
  Response.Write(FArquivoRSS.ReadToEnd);
end;
```

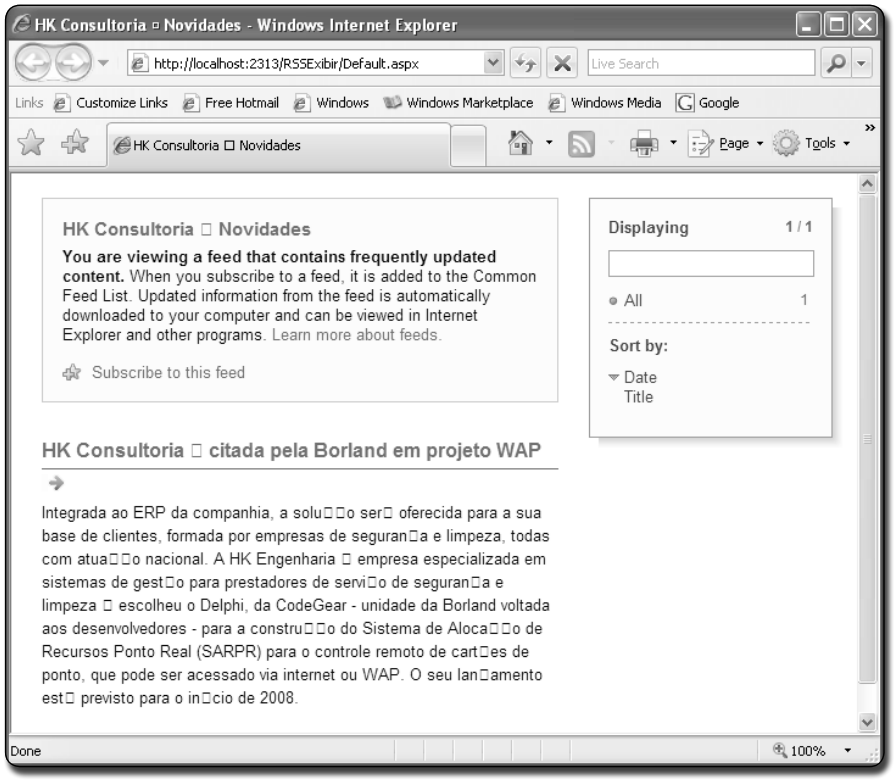


Figura 1. Exibindo o conteúdo do RSS no Internet Explorer

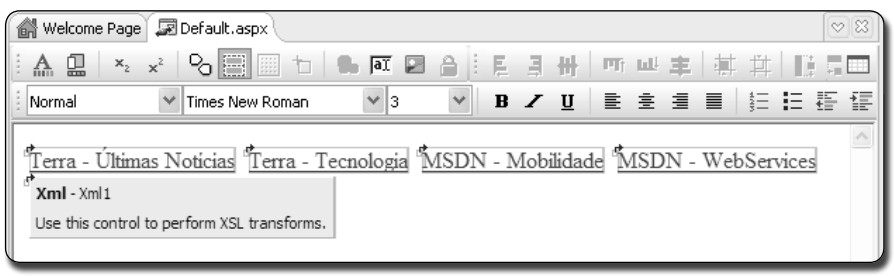


Figura 2. Desenho do WebForm do projeto RSSLeitor

Listagem 3. Código implementado no evento Click do primeiro LinkButton

```
procedure TDefault. lnkbtnTerra_Click(sender: TObject;
  e: System.EventArgs);
var
  FXmlRss:XmlDocument;
begin
  try
    FXmlRss := XmlDocument.Create;
    FXmlRss.Load((Sender as LinkButton).ToolTip);
    FXmlRSS.Document := FXmlRss;
  finally
    FXmlRss.Free;
  end;
end;
```

dll e escolha o item *Add New>Folder* e vamos chamar essa pasta de "MyRSS". Em seguida vamos acessar o menu *File>New>Other>Other Files>Text*. Salve-o como "HKRSS.rss" e adicione o conteúdo da **Listagem 1.** Na seqüência clique com o botão direito na pasta *MyRSS*, escolha a opção *Add* e adicione o arquivo *HKRSS.rss*. Após essa ação o arquivo estará vinculado a nossa pasta.

Em seguida vamos acessar o nosso *WebForm* e dar um duplo clique para acessar o evento *Page_Load* em nossa página e adicionar o código da **Listagem 2.** Nesse código, primeiramente estamos carregando o arquivo RSS em um *Stream*. Então configuramos o *content type* do *browser* para XML e na seqüência exibimos o conteúdo nele. Não podemos esquecer de adicionar a cláusula *Uses* do projeto o *namespace System.IO*. Além disso precisamos acessar o arquivo *.aspx* vinculado ao nosso *WebForm* e excluir a parte HTML, deixando somente a primeira linha, ou seja, nosso *Default.aspx* deverá conter apenas a linha:

```
  <%@ Page language="c#" Debug="true"
    Codebehind="Default.pas"
AutoEventWireup="false"
    Inherits="Default.TDefault" %>
```

Esse conteúdo é excluído, pois estamos enviando para o *browser* um *response* com todo o conteúdo do arquivo RSS para exibição. Após exclusão já podemos executar a aplicação e verificar o resultado conforme **Figura 1**.

Criando um leitor simples para RSS

Seguindo em nosso artigo, agora vamos criar um leitor para arquivos RSS. Crie uma nova aplicação no RAD Studio através do menu *File>New>ASP.NET Web Application – Delphi for .NET*. Salve a aplicação como "RSSLeitor.dproj", mantenha o caminho *default* e o servidor como *Cassini Web Server*. Em seguida, em

nosso *WebForm*, desenhe uma tela como na **Figura 2.** Nessa tela temos quatro componentes *LinkButton*("lnkbtnTerra", "lnkbtnTecnologia", "lnkbtnMobiliade" e "lnkbtnWebServices") da paleta *Web Controls*. Configure a propriedade *Text* de cada um conforme a **Figura 2** e a *ToolTip* modifique para "rss.terra.com.br/0,,EI1,00.xml", "rss.terra.com.br/0,,EI4795,00.xml", "www.msdnbrasil.com.br/rss/movel/rss.xml" e "www.msdnbrasil.com.br/rss/webservices/rss.xml", respectivamente. Por fim adicione um componente XML("xmlRSS") também da paleta *Web Controls*. Em seguida vamos dar um duplo clique no primeiro *LinkButton*("lnkbtnTerra") e implementar o código da **Listagem 3**.

Nesse código, primeiramente estamos carregando o conteúdo do arquivo RSS que será carregado com base na *url* da propriedade *ToolTip* do *LinkButton* em um *XmlDocument* e em seguida atribuímos ele ao componente XML que adicionamos no *design* de nosso *WebForm*. Lembrando que para usar a classe *XmlDocument* é necessário adicionar na seção *Uses* o *namespace System.Xml*. Após implementar o código para o primeiro *LinkButton*, como o mesmo código servirá para todos *LinkButton's*, vamos selecionar os que ainda não foram implementados e no *Object Inspector*. No evento *Click*, faremos um *link* para o código já implementado que aparecerá com o nome *lnkbtnTerra_Click***.** Prosseguindo vamos executar a nossa aplicação pressionando as teclas *Ctrl+Shift+F9* e testar clicando no primeiro *LinkButton*, onde teremos o resultado conforme **Figura 2.**

Como podemos perceber o resultado não foi o esperado, pois como estamos carregando o RSS e exibindo ele diretamente pelo componente XML, o mesmo não está formatado, com isso o resultado visual não foi nada agradável. Para conseguir uma formatação adequada para o arquivo teremos que utilizar um arquivo no formato XSLT. Temos diversos modelos de arquivos XSLT para baixar na internet. Em nosso exemplo acesse o menu *File>New>Other>WebDocuments >XSTL* e clique em *Ok*. Salve o arquivo como "RSSLeitor.xslt" e então o defina

Listagem 4. Definição do arquivo RssLeitor.xslt

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
  "http://www.w3.org/1999/XSL/Transform"
  xmlns:ms="urn:schemas-microsoft-com:xslt"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
<xsl:output method="html" indent="yes"/>
<xsl:param name="TITLE"/>
<xsl:template match="rss">
  <xsl:for-each select="channel/item">
<br>
<strong>
  <a href="{link}" target="_blank">
    <xsl:value-of select="title"
            disable-output-escaping="yes" />
  </a>
</strong>
  (<xsl:value-of select="pubDate" />)
<br>
</br>
<xsl:value-of select="description"
        disable-output-escaping="yes"/>
</br>
<br>
</br>
  </xsl:for-each>
</xsl:template>
<xsl:template match="description">
  <br>
    <xsl:value-of select="."/>
  </br>
</xsl:template>
<xsl:template match="pubDate">
  <xsl:value-of select="formatDate(., 'dd/MM/yyyy')"/>
</xsl:template>
</xsl:stylesheet>
```

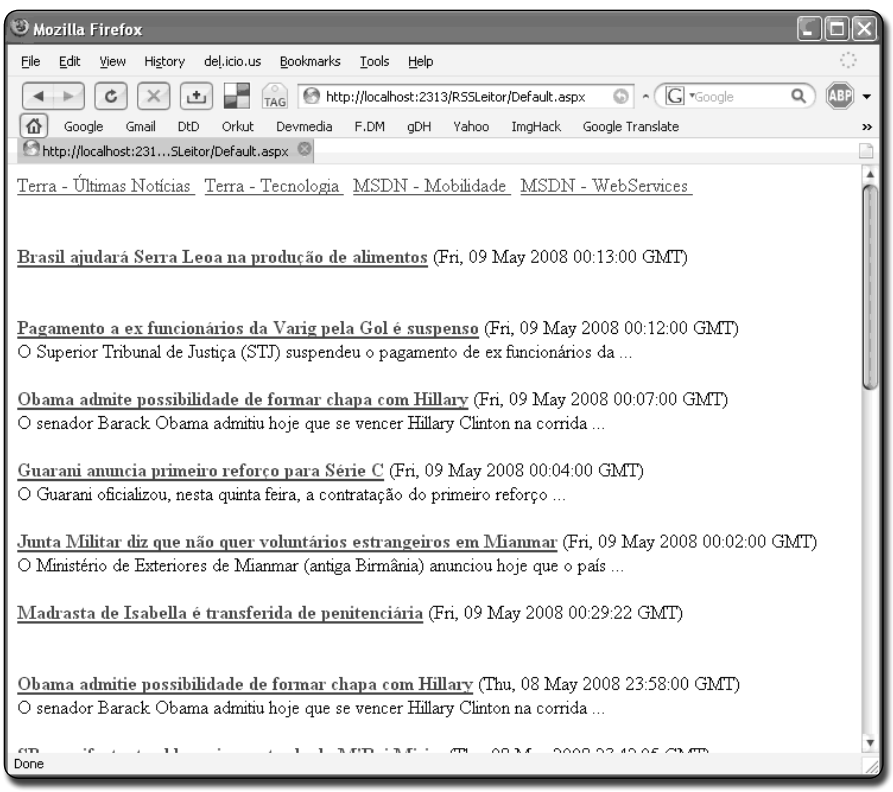


Figura 3. Exibindo o RSS formatado pelo arquivo XSLT

Nota do DevMan

O GridView, no ASP.NET 2.0, substitui o obsoleto DataGrid. O DataGrid que já era um dos mais poderosos controles do ASP.NET, agora possui um controle ainda melhor. O GridView engloba todas as funcionalidades do DataGrid, e ainda pode se ligar diretamente aos novos DataSources do ASP.NET 2.0. Os DataSources são controles especiais que gerenciam em um só lugar toda conexão e obtenção dos dados de um BD, bem como atualizações. Isso significa, por exemplo, que para criar um cadastro com o GridView, você não precisa digitar uma linha de código sequer. Ao contrário do DataGrid, onde uma codificação era necessária, o GridView permite atualização automática dos dados.

conforme a **Listagem 4**. Para fazer com que o nosso componente XML utilize o arquivo XSLT na hora da exibição vamos ter que voltar no código *Click* do *LinkButton* e adicionar a seguinte linha de código conforme segue:

```
xmlRSS.TransformSource :=  'RssLeitor.xslt';
```

Estamos passando ao método *TransformeSource* do componente XML a referência ao XSLT que fará a formatação adequada da notícia. Após essa alteração vamos executar a aplicação e verificar o resultado conforme a **Figura 3.** Basicamente o que o XSLT faz é definir a versão e o formato que o arquivo será usado. Em seguida um laço *for-each* é implementado para que todos os nós, ou seja, todas as notícias sejam impressas na tela. A impressão do link fornece o link em si somando com a data de postagem na notícia.

Criando um leitor com GridView

Continuando o nosso artigo, vamos aprender como utilizar um *GridView* como leitor para exibir as informações vindas de um arquivo RSS. No RAD Studio crie uma nova aplicação através do menu *File>New>ASP.NET Web Application – Delphi for .NET* e salve-o como "GridViewLeitor.dproj". Em nosso *WebForm* inclua um componente *GridView* da paleta *Web Controls* e em seu *Smart Tasks* escolha a opção *Auto Format. Configure* a gosto para deixarmos o visual de nosso *GridView* mais agradável.

Em seguida, através do *Object Inspector*, acesse a propriedade *Columns* e adicione um item chamado *Template Field*. Desmarque a opção *AutoGenerateColumns*, assim geraremos apenas as colunas necessárias. Uma coluna *Template field* é um tipo de coluna que podemos adicionar vários componentes dentro dela e customizar de acordo com a nossa necessidade. Isso é exatamente o que precisamos fazer para exibir o nosso RSS.

Agora clique com o botão direito do mouse sobre o *GridView* para acessar o *PopupMenu* dele e escolher o *Column[0]*. Após escolher esse item vamos ter acesso à edição do *template* e na seção *ItemTemplate* arraste dois *Labels* ("lblTitulo" e "lblDescricao") e um *Hyperlink*("hprLink"). Formate

Figura 4. Adicionando componentes ao Item Template

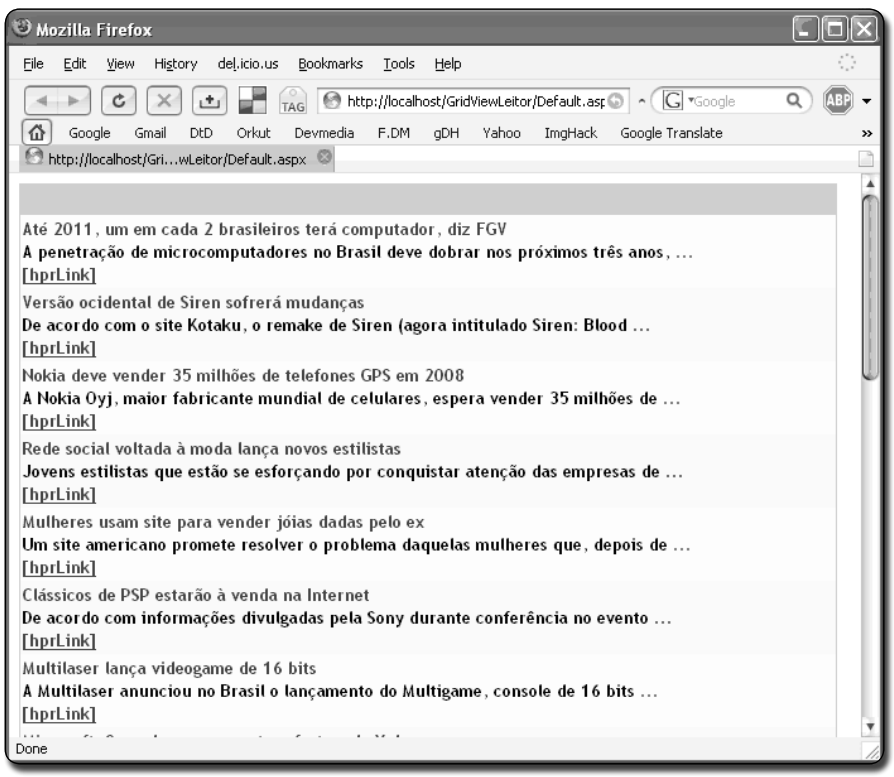


Figura 5. Dados do RSS sendo exibidos no GridView

conforme a **Figura 4** ou de acordo com as cores e tamanhos de fontes de sua preferência. Após configurar, vamos clicar no primeiro *Label* e no seu *Smart Tasks* escolha a opção *Edit DataBindings*. Após selecionar a propriedade *Text* adicione em *Code Expression* a seguinte linha:

```
DataBinder.Eval(Container.DataItem,"title")
```

Repita esse processo com o segundo *Label* ("lblDescricao") desta vez incluindo a expressão:

```
DataBinder.Eval(Container.DataItem,"description")
```

Por fim adicione o código a seguir no *DataBinding* do *hprLink* na propriedade *NavigateUrl:*

```
DataBinder.Eval(Container.DataItem,"link")
```

Todas essas linhas de código são como campos em nosso *GridView*. Perceba que o segundo parâmetro em cada uma delas foi definido como *title, description* e *link*. O *DataBind* faz com que uma propriedade de um campo seja associado a um valor, nesse caso, configuramos os controles de tela de acordo com os campos do RSS.

Para finalizar a edição do *GridView* dê um clique de direita e selecione *End Template Editing*. Clique duas vezes sobre o *WebForm* para acessar o evento *Page_Load* e adicione o código da **Listagem 5**. Nesse código estamos carregando o RSS em um *XmlTextReader* e em seguida carregando ele dentro do nosso *DataSet* (um objeto de memória que guarda um conjunto de dados). Na seqüência passamos o *DataTable* (tabela interna ao *DataSet*), que contém os dados do RSS, para a propriedade *DataSource* do *Grid* e chamamos o método *DataBinding* para preenchimento do *GridView*. Não podemos esquecer de adicionar o *namespace System.Xml* na cláusula *Uses* da nossa *Unit*. Aumente um pouco a largura do *GridView* e execute a aplicação. O resultado vocês poderão ver na **Figura 5**.

Listagem 5. Implementando o Page_Load do WebForm

```
procedure TDefault.Page_Load(sender: System.Object; e: System.EventArgs);
var
  DSLeitor: DataSet;
  XmlLeitor: XmlTextReader;
begin
  try
    XmlLeitor := XmlTextReader.Create('http://rss.terra.com.br/0,,EI4795,00.xml');
    DSLeitor  := DataSet.Create;
    DSLeitor.ReadXml(XmlLeitor);
    DataGrid1.DataSource := DSLeitor.Tables[3];
    DataGrid1.DataBind;
  finally
    DSLeitor.Free;
    XmlLeitor.Free;
  end;
end;
```

Conclusão

Podemos perceber que através do recurso de RSS realmente conseguimos ficar 100% atualizados em relação aos sites de notícias que disponibilizam os arquivos RSS para leitura. Com isso, por meio dos recursos presentes no RAD Studio, conseguimos facilmente incorporar às nossas aplicações listas e mais listas de notícias interessantes aos nossos usuários e ainda mantê-los atualizados sempre que ocorre uma mudança.

Além do recurso visto aqui podemos também gerar os nossos próprios arquivos RSS e gerar toda a estrutura a partir, por exemplo: de uma tabela de notícias presente em um banco de dados.

Certamente em outra edição vamos ensiná-los a implementar esse maravilhoso recurso em nossas aplicações. Espero que tenham gostado do artigo e até a próxima edição. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Controle on-line de vídeo-locadora

Veja como criar um sistema on-line de controle para uma vídeo-locadora - Parte 4

Nesse artigo veremos

- Uso de sessão no ASP.NET;
- Armazenamento de variáveis em sessão;
- Controle de quais usuários se encontram on-line.

Qual a finalidade

- Fazer com que o Web Site determine quais usuários estão atualmente logados.

Quais situações utilizam esses recursos?

- Muitos Web Sites utilizam esse recurso, inclusive fazendo log de ações que determinado usuário logado efetuou.

Continuando o desenvolvimento da nossa aplicação ASP.NET com Delphi for .NET, sobre um sistema on-line de vídeo-locadora, atendendo algumas solicitações que recebi por e-mail, veremos agora como realizar um controle de usuários on-line e logados no sistema, exibindo o resultado em números e também lista de nomes. Mão na massa!

Exibindo usuários logados no sistema

Observados geralmente em fóruns de discussão, a lista de nomes dos usuários logados é uma interessante maneira de saber quais os usuários que estão interagindo com o sistema naquele exato momento. Da mesma forma, o número de usuários ativos em uma aplicação é uma informação importante para o cálculo de estatísticas de acessos a um portal. Mas você deve estar se perguntando, como vou poder controlar todos os usuários que acessam minha página? Em que momento devo contabilizar a visita deste usuário? Como saber se o usuário ainda permanece no sistema? Onde armazenar a lista de usuários que estão on-line? São estas as perguntas que responderei a seguir através de uma implementação prática em nosso projeto, mostrando o passo-a-passo que deverá ser seguido para obter o resultado final.

Entendendo a lógica

O controle de acesso de usuários a uma página é realizado através de eventos que ocorrem de forma automática si-

Maikel Marcelo Scheid
maikelscheid@gmail.com
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

multâneo à solicitação do acesso. Em aplicações ASP.NET temos durante o desenvolvimento a disponibilidade de uma página para que possamos fazer diversas configurações, que geralmente se aplicam a todas as demais páginas do sistema. É nessa página, a "*Global.asax*", que existem métodos que são ativados ao início da aplicação, quando uma sessão de usuário ativo é verificada e métodos que verificam quando um determinado usuário excedeu seu limite de tempo estabelecido na aplicação sem realizar tarefas ou simplesmente deixou o sistema para navegar em outra página e/ou fechou o navegador web.

No Delphi 2006, com o seu projeto de locadora aberto, verifique no *Project Manager* a disponibilidade da "*Global.asax*" e veja em suas declarações de código ("*Global.pas*") os métodos *Application_Start, Session_Start* e *Session_End* referenciados ao início de execução da aplicação, início de uma seção gerada por acesso de usuário e fim de seção de usuários respectivamente. Serão nestes procedimentos que programaremos os códigos para controle e exibição da quantidade e nome dos usuários a navegar pela página.

Para cada acesso, os próprios controles do *.NET Framework* se responsabilizam por criar uma sessão, um ID único para cada usuário. É através desta chave que identificaremos cada um dos usuários. Criada pelo procedimento *Session_Start* a chave permanece inalterada durante todo tempo que o usuário permanecer ativo.

Para usuários ociosos no sistema, estabelecemos um tempo limite de conexão, onde o mesmo permanece conectado e ultrapassando o tempo, sua seção será automaticamente excluída, removendo-o da lista de usuários ativos. Através desta funcionalidade conseguiremos identificar os momentos em que um usuário se conecta, o período que permanece ativo e o momento em que sua sessão é encerrada, deixando o sistema. Mas, onde armazeno estas informações? – Estas informações ficarão armazenadas em uma tabela do banco de dados, onde para cada ação de usuário um novo registro contendo o *ID* da sessão e nome do usuário será gravado. Durante todo o tempo que ele permanecer on-line o registro ficará estável, e ao deixar a aplicação, no evento *Session_End* sua identificação será localizada e excluída do banco de dados.

A tabela a ser criada na base de dados é simples, contendo dois campos apenas (*ID* e Nome do Usuário), onde o código a seguir deverá ser executado através do *Editor de SQL* ligado à base de dados. O código irá criar a tabela "*USUARIOS_ON-LINE*" e dentro da mesma os *Fields ID_USUARIO* e *USUARIO* ambos do tipo *varchar:*

```
CREATE TABLE USUARIOS_ON-LINE (
 ID_USUARIO  VARCHAR(30) CHARACTER SET NONE,
 USUARIO     VARCHAR(60) CHARACTER SET NONE
);
```

Modificando o Global.asax

Criada a tabela no banco de dados, voltaremos ao Delphi onde faremos a codificação dos eventos na página *Global*. Acessando o código (use a tecla de atalho F12), crie a constante "strConexao" na área *Interface* da página informando a *string* de conexão com o banco. Esse valor será usado mais adiante para que o componente *fbConnection* acesse o BD. Da mesma maneira como utilizado no artigo anterior, declare a constante conforme código a seguir. Declare também a variável "User" do tipo *integer* para armazenar e incrementar a quantidade de usuários on-line:

```
var
  Users : Integer;
const
  strConexao = 'User=SYSDBA;
  Password=masterkey;
  Database=
   C:\Inetpub\wwwroot\VideoLocadora\LOCADORA.FDB;';
```

A primeira implementação que faremos será na *procedure Application_Start* onde deverá ser adicionado o seguinte trecho de código no qual estamos atribuindo primeiramente o valor inicial zero na criação da aplicação, e criamos uma sessão de aplicação, atribuindo-lhe também o valor zero. A diferença entre sessão de aplicação (*application*) e sessão de página ("Session") consiste que uma é válida para todos os usuários durante todo tempo da aplicação e a *Session* é valida para cada usuário em uma única sessão por ele criada, obedecendo ainda o tempo limite de conexão fornecido:

```
{ Criando variável e objeto de aplicação
  para contar usuários on-line }
Users := 0;
Application.Add('numberUser','00');
```

O controle e a contagem dos usuários da página faremos também na *Global* adicionando à *procedure Session_Start* o código da **Listagem 1**, com os principais comentários inseridos explicando a funcionalidade e ação a ser realizada.

Observe que no código anterior estamos criando algumas variáveis que serão utilizadas para comunicação e inserção de registros no banco de dados. Na primeira linha do código alteramos também o tempo de sessão para cada usuário ("Timeout") e bloqueamos o código para que não seja acessado por mais de uma pessoa simultaneamente.

Nota do DevMan

As variáveis de sessão ou as sessões em si são amplamente usadas em praticamente todo tipo de site hoje em dia. Sua principal finalidade é guardar informações de usuário para posteriores consultas durante a existência da sessão.

Para quem nunca trabalhou com Internet ou mesmo nunca desenvolveu um site talvez fique confuso o conceito de sessões, mas não é. As sessões funcionam como se fossem variáveis globais em uma aplicação Win32, porém elas "morrem" logo que o site é fechado. Outra característica é que em ASP.NET as páginas ("sites") são stateless, ou seja, toda página .aspx é tratada como se fosse uma nova aplicação sendo inicializada.

Em outras palavras, uma página não enxerga a outra como em formulários diferente em Win32, Bem, então como fazer quando preciso guardar o nome do usuário ou qualquer outro dado importante?

Muito simples: criamos uma sessão e nela passamos a adicionar o conteúdo que precisamos. Quando precisamos resgatar o valor, basta acessarmos a variável de sessão.

Uma boa aplicabilidade para este recurso é quando precisamos listar o carrinho de compras de um internauta em um site. Como saber o que ele comprou para poder gerar a fatura? E se outro usuário comprar os mesmos produtos? Como fica? Bem, aí é que está a razão da utilização de sessões. Se não usarmos sessões cada usuário novo que entrar no sistema fará a leitura e gravação dos dados comprados. Isso significa que se dois ou mais usuários entrarem no site, a probabilidade de entrar produto na cesta de um é muito grande. Por isso as sessões são importantes. Quando utilizamos o código a seguir, estamos informando o e-mail do usuário na sessão EMAIL:

Session['EMAIL'] = 'falecom@adrianosantos.pro.br';

Do mesmo modo, podemos capturar esse valor e retornar em outro Label, veja:

Label1.Text := Session['EMAIL'].ToString();

Isso é importantíssimo, pois cada usuário que entrar terá uma sessão limpa.

Listagem 1. Implementando o código de início da sessão

```
procedure TGlobal.Session_Start(sender: System.Object; e: EventArgs);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prId : FbParameter;
  prUser : FbParameter;
begin
  { Define o tempo de sessão que o usuário poderá ficar ocioso na página }
  Session.Timeout := 20;
  { Bloqueando código para que outros usuários não acessem simultaneamente }
  Application.Lock;
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.InsertCommand := Comand;
  DataAdapter.InsertCommand.Connection := Conn;
  DataAdapter.InsertCommand.CommandText :=
    'insert into usuarios_on-line '+
    '(id_usuario, usuario) values (? , ?)';
  prId := FbParameter.Create;
  prUser := FbParameter.Create;
  DataAdapter.InsertCommand.Parameters.Add(prId);
  DataAdapter.InsertCommand.Parameters.Add(prUser);
  DataAdapter.InsertCommand.Parameters[0].Value := Session.SessionID;
  DataAdapter.InsertCommand.Parameters[1].Value := 'Anônimo';
  DataAdapter.InsertCommand.ExecuteNonQuery;
  { Controle da quantidade de usuários on-line. Adiciona mais 1 à quantidade }
  Users := Convert.ToInt64(Application['numberUser']) + 1;
  Application['numberUser'] := Users.ToString;
  { Liberando bloqueio do código }
  Application.Unlock
end;
```

Listagem 2. Implementando o código de fim da sessão

```
procedure TGlobal.Session_End(sender: System.Object; e: EventArgs);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prId : FbParameter;
begin
  { Bloqueando código para que outros usuários não acessem simultaneamente }
  Application.Lock;
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.DeleteCommand := Comand;
  DataAdapter.DeleteCommand.Connection := Conn;
  DataAdapter.DeleteCommand.CommandText :=
    'delete from usuarios_on-line '+
    'where id_usuario = ?';
  prId := FbParameter.Create;
  DataAdapter.DeleteCommand.Parameters.Add(prId);
  DataAdapter.DeleteCommand.Parameters[0].Value := Session.SessionID;
  DataAdapter.DeleteCommand.ExecuteNonQuery;
  { Controle da quantidade de usuários on-line. Subtrai 1 da quantidade }
  Users := Convert.ToInt64(Application['numberUser']) - 1;
  Application['numberUser'] := Users.ToString;
  { Liberando bloqueio do código }
  Application.Unlock
end;
```

Bloqueado o código, contabilizamos o usuário adicionando a quantidade contida no variável de aplicação e registramos também no banco de dados o ID da sessão e também o nome do usuário, desconhecido até então e por esse motivo guardamos como "Anônimo". Após a ação, o código novamente é desbloqueado.

A liberação de um usuário ocorre de forma semelhante, porém o evento a ser codificado é a *procedure Session_End*, na qual deverá ser adicionado o código da **Listagem 2,** que irá decrementar um registro na quantidade de usuários on-line e também localizar o ID de sessão do usuário na base de dados e excluir o registro.

No código anterior criamos também algumas variáveis para a comunicação com o banco de dados e da mesma forma como anteriormente, bloqueamos o código para prosseguir com a ação de desconectar o usuário. Criamos e configuramos o objeto de conexão com a base de dados e logo após configuramos o item *DeleteCommand* do *DataAdapter* ao qual passamos uma instrução SQL e o parâmetro contendo o ID da sessão que vamos excluir. Realizada a exclusão do usuário, decrementamos 1 (um) usuário da quantidade atual armazenada na variável de aplicação e desbloqueamos o código.

A partir deste momento já ficarão armazenados no banco de dados todos os registros de usuários on-line, lembrando apenas o fato de que todos serão listados como "Anônimos", visando que ainda não escrevemos nenhum código para tratar a identificação em particular de cada usuário, código que veremos a seguir.

Identificação de usuários

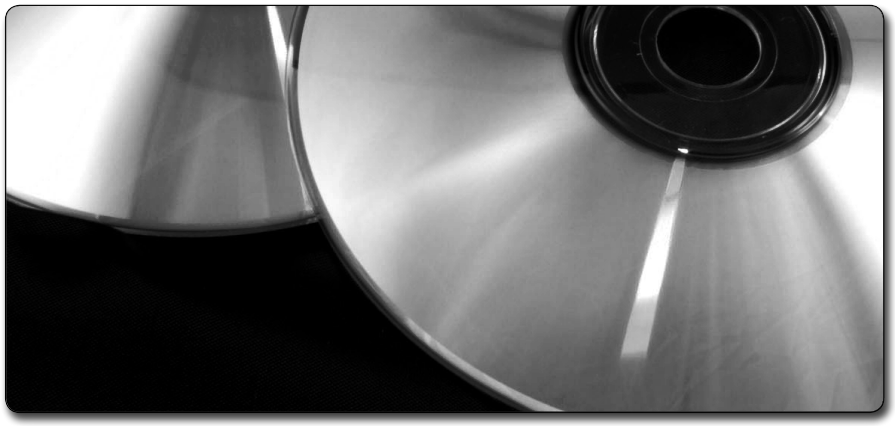
A identificação de usuários ocorre no momento de *login* do mesmo, onde utilizaremos o seu *ID* de sessão ("único para cada usuário") e alteramos o registro no banco de dados, deixando então o usuário de ser "Anônimo" e aparecendo seu nome como um usuário identificado no sistema. Tendo em nossa aplicação duas páginas distintas para o *login* de clientes e também administradores, iremos criar

uma única função centralizando-a em uma *Unit* de códigos. Criaremos nesta *Unit* além da *procedure* de alteração do nome também uma *function* para carregar a lista de todos os usuários ativos no momento. Acesse o menu *File>New>Other>Delphi for .NET Projects>New Files>Unit* ("uUsuariosOn-line.pas") e adicione o código exatamente da forma em que o mesmo se encontra na **Listagem 3.**

Observe que nesta *Unit* de código estamos fazendo referência da *Uses* do Firebird para que possamos utilizar os objetos de comunicação com o banco de dados e após declaramos um procedimento para alteração da identificação do usuário, e uma função que irá resultar em um valor do tipo *string* onde serão listados os nomes dos usuários conectados. Na *procedure* declaramos as variáveis de objetos do Firebird e configuramos através do *DataAdapter* a função *UpdateCommand* a qual recebe dois parâmetros, um contendo o valor da sessão que será localizada na base de dados e outro contendo a identificação do usuário a ser substituída.

Na *function* configuramos também as variáveis de objetos do Firebird e criamos um *SelectCommand* no *DataAdapter* que fará uma pesquisa na identificação de todos os usuários on-line neste exato momento na aplicação. A lista de usuários obtidos através da pesquisa à base de dados será devolvida ao resultado da *function*.

Finalizada a codificação da *Unit* para alteração da identificação e listagem dos usuários on-line, faremos agora sua utilização na página de *login* dos clientes ("login.aspx"), onde logo acima da tabela de área restrita iremos adicionar os textos "Usuários On-line:" e ao lado inserir um *Label* ("lblOn-line") e em uma nova linha o texto "Lista de Usuários:" seguida por um novo *Label*("lblLista"). Com um duplo clique sobre a área da página, acesse o evento *Page_Load* da mesma e adicione o seguinte código, que irá exibir nos *Labels* recém adicionados a quantidade e lista de usuários on-line na página:



Listagem 3. Unit para alteração de identificação e lista de usuários

```
unit uUsuariosOn-line;
interface
uses
 FirebirdSQL.data.Firebird;
procedure AlterarNomeUsuarioOn-line(id, nome: string);
function ListarTodosUsuariosOn-line: String;
implementation
const
  strConexao = 'User=SYSDBA;Password=masterkey;
  Database=C:\Inetpub\wwwroot\VideoLocadora\LOCADORA.FDB;';
procedure AlterarNomeUsuarioOn-line(id, nome: string);
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  prUser : FbParameter;
  prId : FbParameter;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.UpdateCommand := Comand;
  DataAdapter.UpdateCommand.Connection := Conn;
  DataAdapter.UpdateCommand.CommandText :=
    'update usuarios_on-line set '+
    'usuario = ? where id_usuario = ?';
  prUser := FbParameter.Create;
  prId := FbParameter.Create;
  DataAdapter.UpdateCommand.Parameters.Add(prUser);
  DataAdapter.UpdateCommand.Parameters.Add(prId);
  DataAdapter.UpdateCommand.Parameters[0].Value := nome;
  DataAdapter.UpdateCommand.Parameters[1].Value := id;
  if DataAdapter.UpdateCommand.ExecuteNonQuery> 0 then
    ListarTodosUsuariosOn-line;
end;
function ListarTodosUsuariosOn-line:String;
var
  Comand: FbCommand;
  DataAdapter: FbDataAdapter;
  Conn: FbConnection;
  fbReader : FbDataReader;
  Lista : String;
begin
  { Criação dos objetos de conexão }
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  { Atribuição da string de conexão e abertura do BD}
  Conn.ConnectionString := strConexao;
  Conn.Open;
  { Atribuição dos atributos de seleção dos dados }
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText := 'select usuario from usuarios_on-line';
  fbReader := DataAdapter.SelectCommand.ExecuteReader;
  Lista := '';
  while fbReader.Read do
  Result := Result + fbReader['usuario'].ToString+', ';
end;
end.
```

Figura 1. Lista de usuários da página

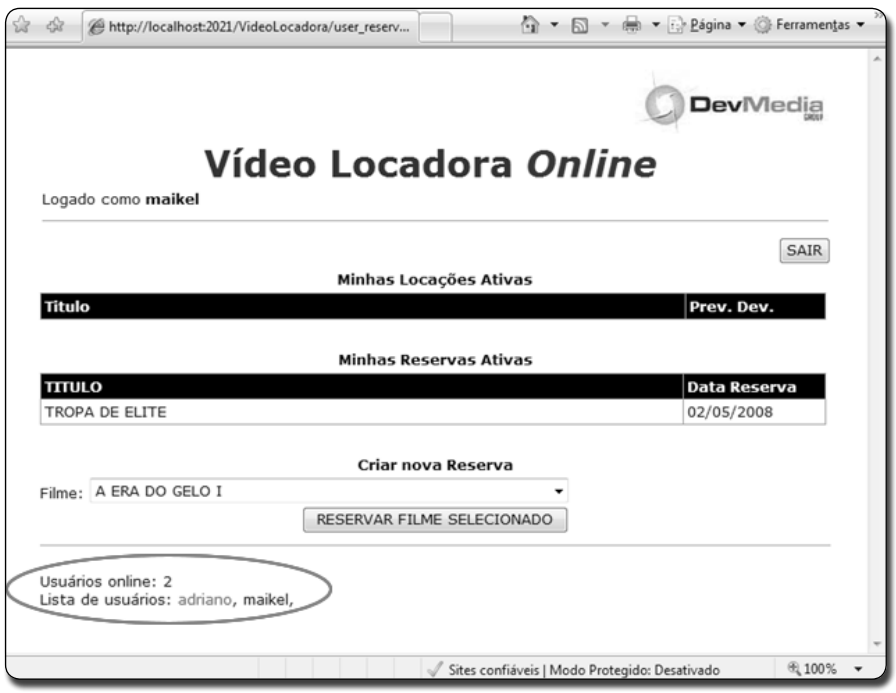
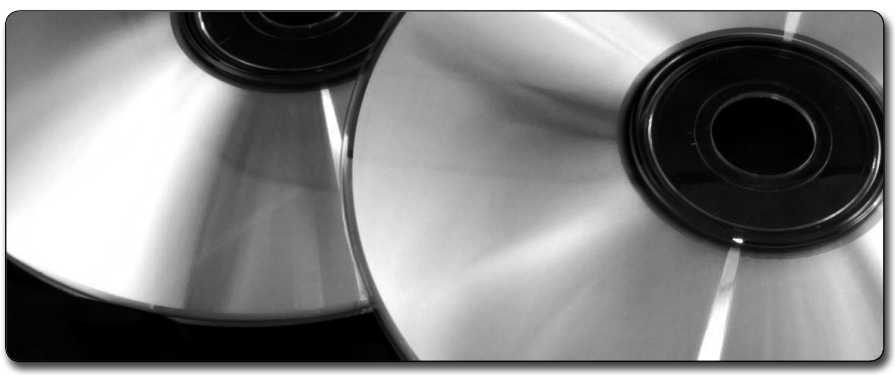


Figura 2. Exibindo quantidade e lista de usuários



```
lblOn-line.Text :=
Application['numberUser'].ToString;
lblLista.Text := ListarTodosUsuariosOn-line;
```

Para alterar o nome do usuário conectado durante o *login*, adicione junto ao código já atribuído ao botão "Login" a seguinte linha de código que irá chamar o procedimento configurado na *Unit* anteriormente criada:

```
[...]
Session['CODIGO']  := fbReader[
  'COD_CLIENTE'].ToString;
AlterarNomeUsuarioOn-line(Session.
SessionID,txtUser.Text);
Response.Redirect('user_reservas.aspx');
[...]
```

Ao executarmos então a página de *login* do sistema, as primeiras informações já serão exibidas na página, onde a quantidade de usuários on-line e os nomes serão listados (**Figura 1**).

Para a página de *login* dos administradores precisamos realizar também a codificação para que o nome do usuário seja alterado, onde iremos adicionar um pequeno trecho HTML para que o nome dos administradores logados, sejam listados na cor vermelha. Acessando a página *Login_adm.aspx* adicione também ao código já existente do botão "*Login*" a seguinte linha de código:

```
[...]
Session['CODIGO']  := fbReader['cod_
usuario'].ToString;
AlterarNomeUsuarioOn-line(Session.
SessionID,'<font color="red">'+txtUser.
Text+'</font>');
Response.Redirect('principal.aspx');
[...]
```

Observe que estamos chamando a *procedure AlterarNomeUsuarioOn-line* e passamos para ela os parâmetros de ID da sessão e também o nome do usuário concatenado ao código HTML que fará a alteração da cor de exibição do *login* de administrador.

Para a listagem da quantidade e lista de usuários on-line nas demais páginas, faremos a configuração de um novo *User Control* o qual utilizaremos como rodapé em todas as páginas. No menu *File>New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET User Control* ("ucrodape.ascx") contendo duas linhas onde deverão ser adicionados os textos "Usuário

On-line:" e "Lista de Usuários" e também deverá adicionar os *Labels* "lblQtde" e "lblLista". Com um duplo clique na área da página acesse o evento *Page_Load* e digite o seguinte código que fará a chamada à variável de aplicação para mostrar a quantidade de usuários e também a chamada a *procedure* para listar os nomes de todos os usuários logados:

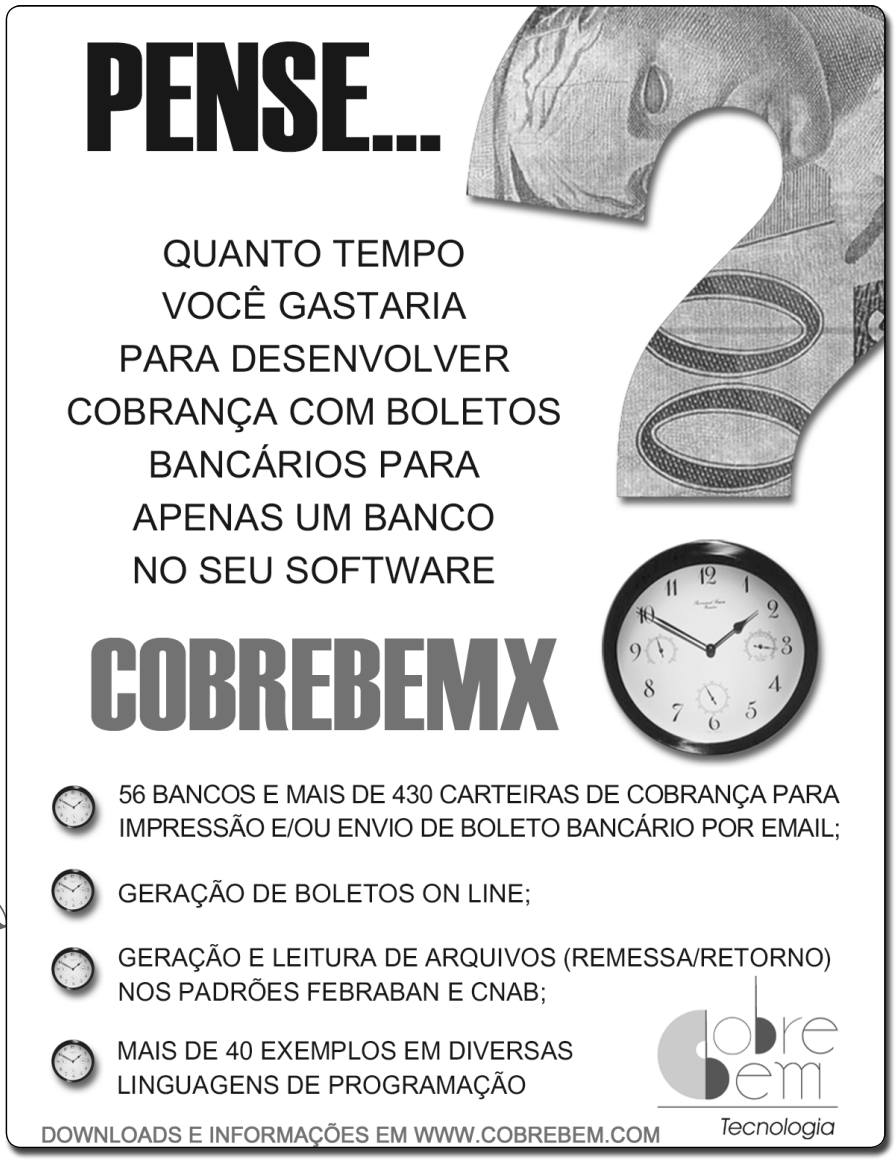
```
lblQtde.Text := Application['numberUser'].
ToString;
lblLista.Text := ListarTodosUsuariosOn-line;
```

Codificada a página de rodapé, abra todas as demais páginas onde deseja adicionar as informações de usuários logados e arraste para dentro delas o objeto *ucrodape.ascx* e faça o teste de funcionamento executando a aplicação, onde o resultado será semelhante ao encontrado na **Figura 2.**

Veja a diferença relacionada entre usuários do tipo cliente e também administradores que se encontram listados em cor vermelha.

Conclusão

Neste artigo vimos a implementação de uma ferramenta bastante interessante para criação de estatísticas de um site, acompanhar os usuários conectados. Neste exemplo estamos apenas exibindo a lista de usuários, mas você poderá implementar também um sistema de *log* onde poderá criar regras e armazenar em banco de dados as ações geradas por cada um dos usuários conectados. Acompanhe no próximo artigo o controle de atrasos em locações de vídeos, aplicando multas de acordo com a quantidade de dias ultrapassados da data de devolução. Grande abraço e até a próxima. ●



Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Dicas de segurança no Firebird

Saiba como trabalhar com usuários/roles e aplicar ou revogar privilégios de acesso à tabelas e/ou campos

Nesse artigo veremos

- Como criar e manipular usuários no banco;
- Criação de permissões;
- Criação e uso de Roles.

Qual a finalidade

- Com a criação de roles podemos determinar o que cada usuário pode fazer no sistema através do banco de dados

Quais situações utilizam esses recursos?

- Em qualquer sistema é comum criarmos políticas de segurança e hierarquias para evitar que usuários comuns cometam erros. São em situações como estas que aplicamos esse tipo de recurso.

Adriano Santos
falecom@adrianosantos.pro.br
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

O Firebird tornou-se, com toda certeza, um dos SGBD's *free* mais utilizados na comunidade Delphi. E não foi à toa que isso aconteceu. Ele é um banco de dados leve, robusto, fácil de gerenciar e o melhor de tudo: grátis. Além dos diversos recursos que o Firebird nos proporciona, não podemos nos esquecer de um fator essencial na escolha de um banco: segurança. Basear-se em desempenho, custo e ferramentas não é tudo se o banco não tiver um nível de segurança aceitável. Nessa edição preparei uma série de dicas para a utilização do sistema de usuários e *roles* do Firebird. Veremos como utilizar o esquema de privilégios, como revogar ("revoke") ou aplicar direitos ("grant") a tabelas e campos. Aprenderemos a usar o aplicativo *GSEC* para criação de usuários e como utilizá-los em nossas aplicações.

Entendendo o GSEC

O Firebird possui uma série de utilitários em sua pasta *bin* no diretório de instalação, tais como: *gbak*, *gfix* e claro o *gsec*. O *gsec* é um utilitário para gerenciamento de usuários e senhas do banco de dados. Com ele conseguimos *incluir*, *excluir* e *modificar* usuários e *roles*. Ele roda por linha de comando localmente ou remotamente e é normalmente utilizado pelo administrador do banco ou DBA. Toda a operação é feita em cima do arquivo *security.fdb*, onde todos os privilégios são guardados. Esse arquivo está situado no diretório de instalação do

SGBD, normalmente em *C:\Arquivos de Programas\Firebird\Firebird_Versao*. Vamos a uma visão geral do aplicativo.

> **Nota**: A partir da versão 2.0 do Firebird, o arquivo de segurança do banco passou a chamar-se de *security2.fdb* para diferenciar das versões abaixo de 2.0. Espera-se para a vesão 3.0, denominada Vulcan, que o *security* não seja mais um arquivo separado e sim parte do banco de dados.

Comandos do GSEC

A inclusão de usuários, bem como a definição de sua senha, é feita utilizando-se os comandos *add <nome_usuario>* e *–pw*, respectivamente. Veja um exemplo a seguir:

```
GSEC>add ADRIANO -pw 123456
```

Em seguida, para termos certeza de que o usuário foi adicionado com sucesso, podemos usar o comando *display* que mostra todos os usuários do banco, como a seguir (**Figura 1):**

```
GSEC>display
```

Como podemos notar na **Figura 1**, temos como usuários além do SYSDBA os nomes *ADRIANO, JOSE, GUINTHER* e *JOANA*. Note também que o *prompt* está diferente apresentando *GSEC>* ao invés do convencional *C:*, isso porque estamos acessando a área de administração do Firebird. Com o Firebird instalado em seu computador, abra uma janela de *prompt* do DOS e entre na pasta *bin* do Firebird. No meu caso fica situada em *C:\Program Files\Firebird\Firebird_2_1\bin*.

Perceba que já estou usando a versão 2.1.0 RC2, ainda em beta teste. Ao entrar no diretório digite a linha de comando em seguida:

```
C:\Program Files\Firebird\Firebird_2_1\
bin>gsec -user SYSDBA -pw masterkey
```

Esse comando é usado para logar-se no banco de dados e ter acesso a todos os seus recursos. Perceba que o *prompt* do DOS muda para *GSEC>* como mencionado anteriormente e agora já temos condições de executar nossos testes. Crie um novo usuário usando a linha de comando que vimos anteriormente, ou seja:

```
GSEC>add ADRIANO -pw 123456
```

Em seguida digite *display* e tecle *enter*. Facilmente notamos que o usuário ADRIANO foi incluído com sucesso. Para testarmos se ele tem acesso ao SGBD vamos sair do banco usando o comando *quit*. Em seguida tente conectar-se usando o comando a seguir:

```
GSEC>gsec -user ADRIANO -password 123456
```

Se o *prompt* de comando virar para *GSEC>* é porque temos total acesso ao servidor. Retorne ao usuário SYSDBA e vamos fazer um teste de exclusão. Para excluir um usuário basta usar o comando *delete* conforme segue:

```
GSEC>delete ADRIANO
```

Como podemos ver a inclusão e exclusão de usuários é bem simples. Vejamos agora como modificar as configurações

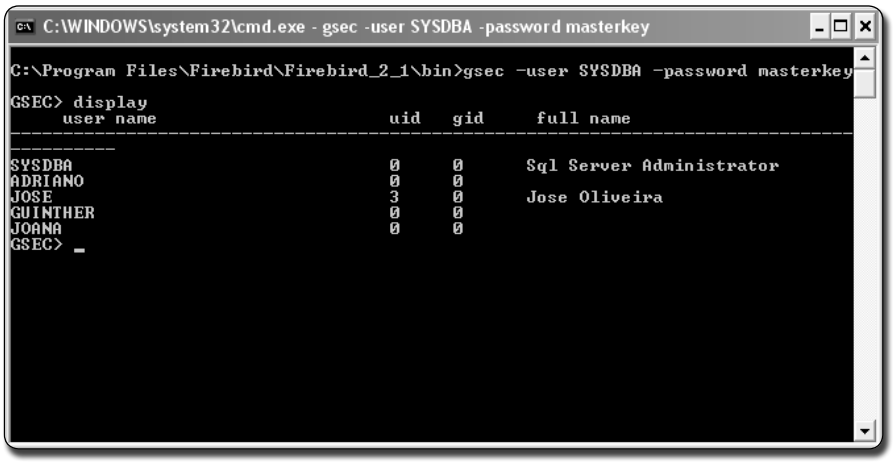


Figura 1. Todos os usuários criados no banco

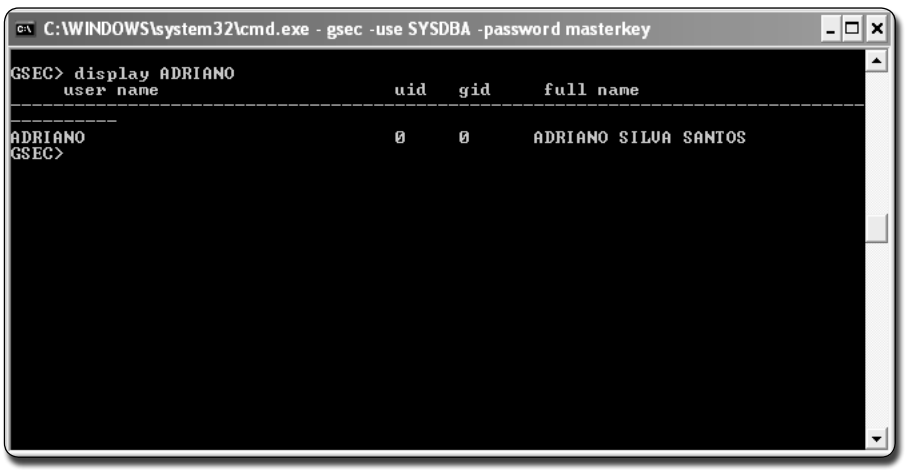


Figura 2. Mostrando as informações de ADRIANO

Nota do DevMan

Atualmente o código do Vulcan é mantido pelo Jim Starkley e está sendo embutido no novo Firebird. Entre as metas para o novo Firebird estão:

Portabilidade: O Vulcan foi desenvolvido simultaneamente para quatro plataformas que são: 32 bits usando o compilador Microsoft Visual Studio 7, 32 e 64 bits, Linux para AMD64 usando várias versões do GCC ("Compilador Linux") e 64 bits para Solaris usando o compilador Sun Forte C++;

Performance: Maior performance em vista do Firebird 1.5. A equipe da FirebirdSQL tem como meta aumentar ainda mais a velocidade de acesso e processamento do Firebird;

Segurança: Integração do security.fdb ao banco de dados. Isso siginifica que não existirá mais um arquivo separado para guardar senhas do usuário SYSDBA e demais, tornando assim ainda mais seguro o SGBD;

Para melhor entender todo o desenvolvimento dessa nova versão aconselho acessar o link www.ibphoenix.com/main.nfs?page=vul_development que contém todas as informações necessárias incluindo um documento em PDF sobre o Overview do Vulcan.

Listagem 1. Estrutura de criação do banco e tabelas

```
SET SQL DIALECT 3;
SET NAMES WIN1252;

CREATE DATABASE '<Caminho>\Seguranca.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252;

CREATE GENERATOR GEN_CLIENTES_ID;
SET GENERATOR GEN_CLIENTES_ID TO 3;

CREATE TABLE CLIENTES (
    ID        INTEGER NOT NULL,
    NOME      VARCHAR(40),
    ENDERECO  VARCHAR(200),
    CIDADE    VARCHAR(40),
    ESTADO    VARCHAR(2),
    CEP       VARCHAR(9)
);
ALTER TABLE CLIENTES ADD CONSTRAINT PK_CLIENTES
  PRIMARY KEY (ID);
SET TERM ^ ;
/* Trigger: CLIENTES_BI */
CREATE TRIGGER CLIENTES_BI FOR CLIENTES
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = GEN_ID(GEN_CLIENTES_ID,1);
END
^
SET TERM ; ^
```

Listagem 2. Código de conexão com o banco

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if sqlConexao.Connected then
    sqlConexao.Connected := False;

  with sqlConexao.Params do
  begin
    Clear;
    Add('DriverName=Interbase');
    Add('Database=<Caminho>\SEGURANCA.FDB');
    Add('RoleName=');
    Add('User_Name=' + Edit1.Text);
    Add('Password=' + Edit2.Text);
    Add('ServerCharSet=');
    Add('SQLDialect=3');
    Add('BlobSize=-1');
    Add('CommitRetain=False');
    Add('WaitOnLocks=True');
    Add('ErrorResourceFile=');
    Add('LocaleCode=0000');
    Add('Interbase TransIsolation=ReadCommited');
    Add('Trim Char=False');
  end;

  sqlConexao.Connected := True;
  if sqlConexao.Connected then
    cdsClientes.Active := True;
end;
```

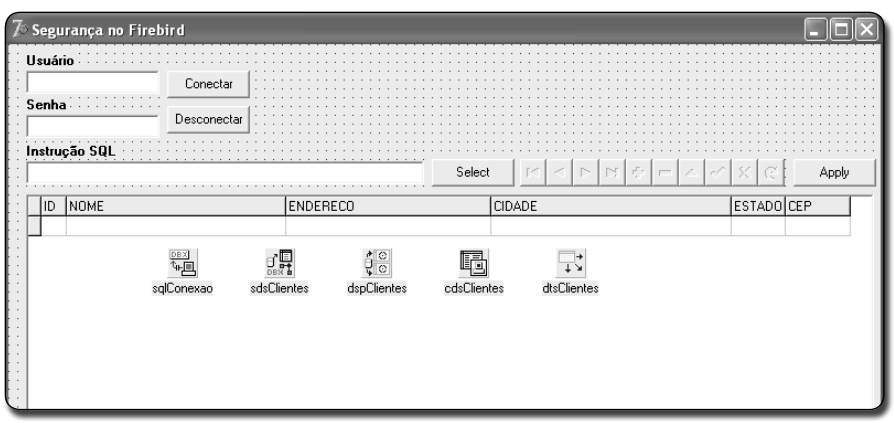


Figura 3. Tela exemplo para conexão com o banco

de conta de um usuário usando o comando *modify*. Todos os usuários do banco de dados podem ter *fname* ("primero nome"), *mname* ("nome do meio") ou *lname* ("último nome"). Vamos aprender agora como modificar esses parâmetros. Para o usuário ADRIANO vamos adicionar o primeiro, segundo e último nome. Volte ao *prompt* e digite:

```
GSEC>modify ADRIANO -fname ADRIANO -mname
SILVA -lname SANTOS
```

Em seguida podemos invocar o comando *display* seguido do nome do usuário, como podemos ver em seguida. O *GSEC* nos retornará uma tela como a **Figura 2**. Veja um fato interessante, o último campo mostrado pelo *display* é uma concatenação dos três parâmetros anteriores, portanto a tabela de usuários do Firebird possui os campos *FIRST_NAME*, *MIDDLE_NAME*, *LAST_NAME* e *FULL_NAME*:

```
GSEC>display ADRIANO
```

A modificação de senha de usuário segue o mesmo esquema anterior. Basta usarmos o comando *display* juntamente com o parâmetro *–pw*, veja:

```
GSEC>display ADRIANO -pw 456
```

No exemplo anterior estamos modificando a senha do usuário *ADRIANO* para *456*.

Conectando-se ao banco pela aplicação

A grande maioria dos desenvolvedores costuma utilizar o usuário e senha do Firebird (usuário: "SYSDBA" e senha: "masterkey") para conectar sua aplicação ao banco de dados, o que não é aconselhável. O correto seria utilizar os usuários criados diretamente no SGBD e dar privilégios e acesso a cada um deles ou a grupos, os *roles*. Nessa seção veremos como fazer isso utilizando os componentes *DBExpress*.

Antes de mais nada criaremos um banco de dados para efetuarmos nossos testes. Para isso utilize seu gerenciador preferido para trabalhar com SGBD's Firebird e crie um arquivo com o nome "Seguranca.fdb" e nele adicione uma tabela de clientes conforme o script da **Listagem 1**. Inclua alguns registros nela

para que possamos efetuar os testes.

Abra o Delphi e crie um nova aplicação salvando-a como "Seguranca.dpr". Na tela principal inclua um componente *SQLConnection*("sqlConexao") da paleta *dbExpress*. Clique duas vezes no componente e configure uma nova conexão de maneira que se conecte a nossa base de dados criada anteriormente. Em seguida inclua um *SQLDataSet*("sdsClientes") conectado ao *sqlConexao*, um *DataSetProvider*("dspClientes") conectado ao *sdsClientes* pela propriedade *DataSet* e por fim um *ClientDataSet*("cdsClientes") ligado ao *dspClientes*. Na propriedade *CommandText* do *sdsClientes* digite uma instrução SQL que traga todos os dados da tabela *Clientes*, ex.:

```
SELECT * FROM CLIENTES
```

Acesse o *Fields Editor* do *cdsClientes* e com o botão direito adicione todos os campos da tabela. Para que possamos ver os dados na tela inclua também um *DataSource*("dtsClientes") e um *DBGrid*("grdCliente"), ambos ligados aos seus respectivos componentes.

Inclua também dois *Edits*("edtUsuario" e "edtSenha") para usuário/senha e outro para instruções SQL("edtSQL"). Devemos colocar mais alguns botões para conectar e desconectar, para aplicar instruções SQL e por fim um *Button* para efetuar *ApplyUpdates* quando necessário (**Figura 3).** Se preferir insira um *DBNavigator* e vincule-o ao *dtsClientes*.

A primeira coisa que veremos a seguir é "como fazer a conexão de nossa aplicação com o banco de dados usando um usuário específico". Basicamente o que precisamos fazer é alterar em *runtime* os parâmetros *User_Name* e *Password* do *sqlConexao*. E já que faremos isso em *runtime*, aproveitamos para limpar todos os parâmetros e renová-los com nosso padrão. Digite o código da **Listagem 2** no evento *OnClick* do botão *Conectar*.

Observe que estamos utilizando o conteúdo dos campos *edtUsuario* e *edtSenha* para informar ao *sqlConexao* qual usuário e senha, respectivamente, serão usados para conectar-se ao *Seguranca.fdb*. A programação em si é muito simples de entender. Estamos apenas confirmando se o *sqlConexao* está conectado e desconectando-o para que possamos limpar seus parâmetros. Em seguida limpamos e adicionamos novamente seus parâmetros, desta vez informando dinamicamente usuário e senha como já foi explicado anteriormente. Por fim conectamos ao banco e caso ocorra tudo bem abrimos o *ClientDataSet* exibindo os registros da tabela *Clientes*.

Todas as configurações estando corretas podemos executar a aplicação e testar. Certamente os registros da tabela *Clientes* retornarão no *DBGrid* assim que conectarmos ao banco (**Figura 4**).

Trabalhando com privilégios

Quando conectamos utilizando o usuário SYSDBA, ou seja, o usuário principal também conhecido como *Administrador*, estamos dando plenos poderes ao sistema para realizar qualquer tipo de operação no SGBD, incluindo execução de *Stored Procedures* e *Triggers*. Uma grande maioria de desenvolvedores utiliza desta forma, ou seja, não se preocupa em mudar ou mesmo criar sua própria senha de administrador. Por esse motivo, acaba utilizando por padrão a senha que o próprio banco fornece. Veja na **Listagem 3** como o *SQLConnection* é configurado por *default* logo quando criamos uma conexão nova. O que podemos ver na listagem é a propriedade *Params* do *sqlConexao*. Perceba que nos parâmetros *User_Name* e *Password* estamos usando *SYSDBA* e *masterkey*, respectivamente.

Observe também o parâmetro *RoleName*. Ele possui o valor *RoleName*, ou seja, seu *default*. Nesse ponto estamos utilizando os privilégios do administrador. Mas podemos facilmente mudar esse cenário e fazer com que o sistema utilize os privilégios de cada usuário criado no banco. Vejamos como isso funciona.

Primeiramente faremos uso dos usuários do *SGBD* sem a utilização de *roles*, em seguida veremos o que exatamente são *roles* e como usá-los. Para este artigo usaremos o IBExpert para realizar as operações de manutenção no banco. Portanto no IBExpert abra uma nova janela de *scripts* usando o

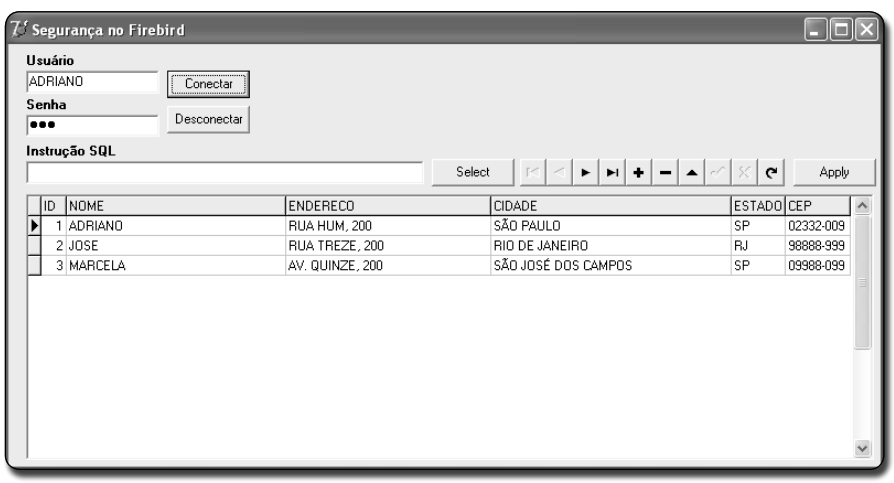


Figura 4. Banco de dados conectado usando usuário diferente do SYSDBA

Listagem 3. Propriedade Params default

```
DriverName=Interbase
Database=<Caminho>\Seguranca.fdb
RoleName=RoleName
User_Name=SYSDBA
Password=masterkey
ServerCharSet=
SQLDialect=3
BlobSize=-1
CommitRetain=False
WaitOnLocks=True
ErrorResourceFile=
LocaleCode=0000
Interbase TransIsolation=ReadCommited
Trim Char=False
```

menu *SQL Editor* ou pressione *F12*. Usaremos o comando *Grant* para adicionar privilégios e *Revoke* para removê-los. Retorne ao *prompt* do *GSEC* se necessário e inclua novos usuários. Aqui utilizarei os usuários *Adriano* e *Guinther*. Cada um terá seus próprios privilégios.

O usuário *Adriano* terá acesso somente à execução de *Select's*, ou seja, ele não poderá *incluir, excluir* ou *alterar* registros em determinada tabela. Já o usuário *Guinther*, terá acesso total.

A sintaxe do comando *Grant* é:

```
grant [select | insert | delete | update |
references | execute | all]
on [table tabela(s)] to [usuario(s) |
role(s) | procedures(s) | public]
[with [ grant | admin ] option]
```

A explicação para cada item é a seguinte:

• *Select*: Permite que usuário execute comandos de seleção de registros;

• *Insert, Delete* e *Update*: Como o próprio parâmetro sugere, significa privilégios para *inclusão, exclusão* e *alteração* de registros;

• *References*: Permite definir chaves estrangeiras;

• *All*: Acesso total, incluindo execução de *Stored Procedures* e *Triggers*;

Em seguida informamos no *Grant* para qual(is) tabela(s) os privilégios são válidos ("on table(s)") e qual(is) usuário(s), *role*(s) e/ou *procedure(s)* serão aplicados. Para entender um pouco melhor, vamos fazer um pequeno teste. Digite o *script* seguinte, execute e confirme clicando no botão *Commit Transaction* do IBExpert:

```
GRANT SELECT ON CLIENTES TO ADRIANO;
```

O *script* anterior aplica o privilégio apenas para execução de *Select* na tabela CLIENTES para o usuário ADRIANO. Execute novamente a aplicação e experimente conectar-se ao banco de dados usando o usuário ADRIANO. Certamente a conexão e a seleção dos dados serão feitas normalmente, porém se tentarmos alterar um registro, gravar e efetuar um *ApplyUpdates*, veremos que uma exceção será levantada e a operação não será confirmada. Para testarmos, faça uma chamada ao método *ApplyUpdates* em um dos *Buttons* da tela e faça novamente o teste, desta vez simulando a alteração de um registro qualquer. Veja a exceção gerada na **Figura 5**.

A mensagem gerada pelo Delphi, como podemos perceber, foi "no permission for update/write access to TABLE CLIENTES". Isso porque o usuário ADRIANO não possui permissão para escrita na tabela CLIENTES. O erro só será notado ao executar o botão que possui a chamada ao método *ApplyUpdates* devido ao fato de estarmos usando a estrutura de acesso bidirecional, ou seja, *SQLConnection > SQLDataSet > DataSetProvider > ClientDataSet*. Em outras palavras, o *Post* executado no botão *Save* do *DBNavigator* ou mesmo se tivéssemos um botão separado para isso, não geraria tal exceção, já que os dados estão carregados na memória ("ClientDataSet") e somente são enviados ao banco por meio do *ApplyUpdates*.

Façamos um novo teste agora. Vamos dar permissão total ao usuário *Guinther* e tentar fazer alterações no banco. Para isso, volte ao IBExpert e digite o comando a seguir:

```
GRANT ALL ON CLIENTES TO GUINTHER;
```

Execute o *script*, aplique um *commit* e logo em seguida abra nossa aplicação teste. Para testarmos novamente, utilize o usuário GUINTHER. Logo perceberá que o erro não ocorre. O parâmetro ALL significa que estamos dando total acesso ao usuário, por isso ele poderá selecionar ("Select"), incluir ("Insert"), excluir ("Delete"), atualizar ("Update") além de executar *Stored Procedures* e *Triggers*.

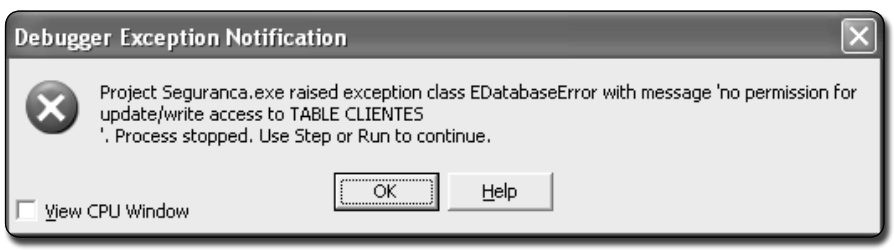


Figura 5. Exceção gerada pelo ApplyUpdates

Figura 6. Mensagem de erro no ReconcileError

Revogando privilégios

O *Revoke* é um comando contrário ao *Grant*, ou seja, ele revoga os direitos que o usuário tem sobre determinada *tabela, view* etc. Sua sintaxe é bem simples:

```
revoke [select | insert | delete | update |
references | execute | all]
on [table tabela(s)] from [usuario(s) |
role(s) | procedures(s) | public]
```

A explicação para cada parâmetro é a mesma encontrada no comando *Grant*, porém com sentido contrário como já explicado. Em outras palavras quando usamos *revoke*, estamos retirando o direito do usuário sobre tal operação. Ex.:

```
REVOKE DELETE ON CLIENTES FROM GUINTHER;
```

Estamos retirando o direito do usuário GUINTHER de fazer exclusões no banco de dados. Rode a aplicação novamente e faça o teste. Você receberá como exceção a mensagem que vimos ao tentarmos alterar um registro com o usuário ADRIANO. Contudo, o erro só nos aparece se executarmos um *ApplyUpdates* e

se estivermos executando de dentro do IDE do Delphi. Esse último caso é mais complicado, pois se estivermos rodando a aplicação por fora do Delphi o erro não será exibido e o registro não excluído. Faremos uma pequena modificação, para que possamos saber quando ocorreu algum erro. No evento *OnReconcileError* do *cdsClientes* inclua o código a seguir:

```
MessageDlg('Ocorreu um erro ao salvar no
           banco de dados.' + #13 +
          'Erro original:' + #13#13 +
           E.Message, mtError, [mbOk], 0);
```

Estamos interceptando o erro e mostrando-o em uma *MessageDlg* (**Figura 6**). Sendo assim, nossa mensagem de erro torna-se personalizada.

Trabalhando com Roles

A primeira pergunta é: O que são *Roles*? *Role* é um grupo, onde podemos definir privilégios a ele e em seguida atribuí-lo a um ou mais usuários. A principal vantagem em se usar *Role*, é que você define apenas uma vez a que cada usuário terá ou não acesso. Em um sistema com muitos funcionários acessando o software, a criação de grupos pode ser muito benéfica, pois poupa muito tempo. Sua utilização é muito simples.

Começaremos pela criação de *Roles*. Para criar uma *role* no Firebird, basta executar o comando *create role* juntamente com o nome desse grupo. Ex.:

```
CREATE ROLE GRUPO_SELECT;
CRATE ROLE GRUPO_ALTERACAO;
```

Como podemos ver estamos criando dois grupos. O próximo passo é especificar o que cada grupo terá direito a acessar. Assim como os nomes dos grupos sugerem vamos fazer com que a role GRUPO_SELECT não tenha acesso à *inclusão, alteração, exclusão* e execução de *Stored Procedures* e *Triggers*. Para isso digite e execute cada comando a seguir:

```
GRANT SELECT ON CLIENTES TO GRUPO_SELECT;
GRANT ALL ON CLIENTES TO GRUPO_ALTERACAO;
```

Veja que usamos o mesmo comando *Grant* usado anteriormente, só que desta vez atribuímos seus privilégios diretamente a *role*. Em resumo estamos configurando o grupo GRUPO_SELECT para executar somente *Select* no banco, diferente do GRUPO_ALTERACAO que tem acesso total ("ALL").

O último passo é alocar os usuários em seus respectivos grupos:

```
GRANT GRUPO_SELECT TO ADRIANO;
GRANT GRUPO_ALTERACAO TO GUINTHER;
```

Agora fica ainda mais fácil de entender. Especificamos que o usuário ADRIANO é do GRUPO_SELECT e só tem acesso à consulta. Diferente do usuário GUINTHER que é do GRUPO_ALTERACAO e possui acesso total às rotinas do SGBD. Experimente executar a aplicação e fazer os devidos testes. Certamente terá uma surpresa, ambos usuários não possuem acesso ao sistema devido a mensagem de erro "no permission for update/write access to TABLE CLIENTES". Isso acontece porque não informamos ao *SQLConnection* qual a *role* estará sendo usada. Vamos rever o código da **Listagem 3**. Note que no parâmetro *RoleName* estamos mantendo seu valor padrão, ou seja, *RoleName=RoleName*. Teremos que mudar esse procedimento já que o sistema interpreta que nenhuma *role* está sendo usada no momento.

Insira um novo botão na tela e mude seu *Caption* para "Conectar usando Role" e logo em seguida clique duas vezes nele

Listagem 4. Código do botão Conectar usando Role

```
procedure TForm1.ConectarUsandoRole5Click(Sender:
  TObject);
var
  Role : string;
  procedure AtualizarParametros(AUser, ASenha,
    ARole: string);
  begin
    with sqlConexao.Params do
    begin
      Clear;
      Add('DriverName=Interbase');
      Add('Database=<Caminho>\SEGURANCA.FDB');
      Add('RoleName=' + ARole);
      Add('User_Name=' + AUser);
      Add('Password=' + ASenha);
      Add('ServerCharSet=');
      Add('SQLDialect=1');
      Add('BlobSize=-1');
      Add('CommitRetain=False');
      Add('WaitOnLocks=True');
      Add('ErrorResourceFile=');
      Add('LocaleCode=0000');
      Add('Interbase TransIsolation=ReadCommited');
      Add('Trim Char=False');
    end;
  end;
begin
  if sqlConexao.Connected then
    sqlConexao.Connected := False;
  AtualizarParametros('SYSDBA', 'masterkey', '');
  sqlConexao.Connected := True;

  sdsClientes.Close;
  sdsClientes.CommandText := '';
  sdsClientes.CommandText := 'SELECT RDB$RELATION_NAME
    AS ROLE FROM RDB$USER_PRIVILEGES WHERE
    RDB$USER = ' + QuotedStr(Edit1.Text);
  cdsAuxiliar.Params.Clear;
  cdsAuxiliar.FetchParams;
  cdsAuxiliar.Open;

  Role := cdsAuxiliar.FieldByName('ROLE').AsString;

  sqlConexao.Connected := False;

  AtualizarParametros(Edit1.Text, Edit2.Text, Role);
  sqlConexao.Connected := True;

  if sqlConexao.Connected then
  begin
    sdsClientes.CommandText := '';
    sdsClientes.CommandText :='SELECT * FROM
      CLIENTES';
    cdsClientes.Params.Clear;
    cdsClientes.FetchParams;
    cdsClientes.Open;
  end;
end;
```

para visualizarmos a página de código e seu evento *OnClick*. No evento mencionado digite o código da **Listagem 4**.

Vamos ao que interessa. O segredo de tudo está no parâmetro *RoleName*. Nele precisamos informar "qual o role será usado", do contrário o banco retornará que tal usuário não tem acesso a determinadas operações.

Mas para que isso se concretize precisamos saber qual o *Role*, ou seja, a qual grupo o usuário faz parte. E como saber essa informação? A resposta é simples: recorrendo às tabelas de sistema.

O Firebird assim como outros bancos de dados possui algumas tabelas adicionais que chamamos de *System Tables*, ou *Tabelas de Sistema*. Uma série de informações referentes ao banco de dados em questão são mostradas/armazenadas nessas tabelas. No Firebird elas começam pelo prefixo *RDB$*. Faremos uso delas para detectar qual a *role* em que determinado usuário se encontra alocado. A *System Table* que vamos usar é a *RDB$USER_PRIVILEGIES* que guarda os privilégios.

A explicação dessa tabela é bem simples: cada vez que usamos o comando *grant* ou *revoke* para dar ou retirar privilégios de um usuário ou *role*, essa tabela é atualizada. Execute a *Select* a seguir e veja o que ocorre:

```
SELECT * FROM RDB$PRIVILEGIES
```

Notará que cada privilégio de cada *usuário/role* é listado, e assim fica fácil entendermos o que vamos fazer. Faremos uma *Select* nessa fonte de dados para que possamos descobrir qual o grupo que determinado usuário está alocado, mas para isso precisaremos de um novo *ClientDataSet*("cdsAuxiliar"). Insira-o em tela e vincule-o ao *dspClientes*. A **Listagem 4** fará toda a diferença. Estamos conectando ao banco de dados usando o usuário *SYSDBA* e senha *masterkey* e listando o campo RDB$RELATION_NAME com o apelido ROLE da tabela RDB$USER_PRIVILEGIES. Por fim pegamos esse resultado e usamos como parâmetro na função *AtualizarParametros*. Em *AtualizarParametros* substituímos o parâmetro *RoleName* por *RoleName=ARole*, ou seja, a *Role* recebida no parâmetro da *procedure*.

O *cdsAuxiliar* é usado como *ClientDataSet* de apoio, pra poder receber o resultado da *Select* na tabela de sistema. Se rodar o sistema, perceberá que temos o mesmo cenário usado nos últimos exemplos, porém desta vez definimos os privilégios para o grupo e não para cada usuário. Para ter certeza de que o sistema está usando os privilégios configurados nas *Roles* GRUPO_SELECT e GRUPO_ALTERACAO, rode os *scripts* em seguida e execute novamente o sistema fazendo os mesmos testes. Os *scripts* revogam todos os direitos de cada usuário:

```
REVOKE ALL ON CLIENTES TO ADRIANO;
REVOKE ALL ON CLIENTES TO GUINTHER;
```

Gerenciando usuários pelo IBExpert

Para finalizar nosso artigo vejamos como fazer todo esse gerenciamento usando o IBExpert. O IBExpert em sua versão *free* ("Personal Edition"), permite apenas a inclusão, alteração e exclusão de usuários e *roles*, bem como atribuição destes *roles* a usuários (**Figura 7**).

A tela de gerenciamento pode ser acessada pelo menu *Tools|User Manager*. Na primeira aba, *Users*, podemos adicionar ("Add"), alterar ("Edit") e excluir ("Delete") um usuário. A aba seguinte, *Roles*, é destinada a inclusão e exclusão. E por fim, *Membership*, é onde atribuímos um grupo a determinado usuário. Para isso basta marcarmos com um "x" na coluna "G" o grupo que desejamos atribuir.

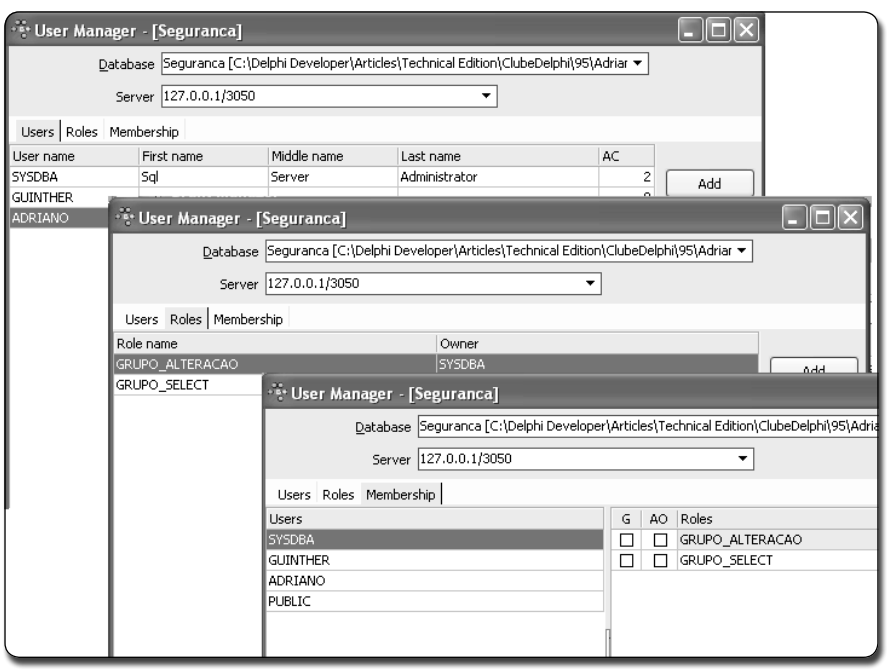


Figura 7. IBExpert, inclusão e exclusão de usuários e roles

O grande trunfo do IBExpert está em sua versão *full*, completa. Com ela podemos atribuir ou revogar direitos a grupos e usuários de forma visual. Veja na **Figura 8** que o GRUPO_ALTERACAO possui permissão total ("ícones em verde limão") em todas as colunas. Note também que podemos dar direitos ou revogá-los para cada campo da tabela. Isso significa que podemos ter um nível de segurança ainda mais avançado, não permitindo que determinado grupo ou usuário possa efetuar atualizações em um ou mais campos da tabela em questão.

À esquerda da janela encontramos um *ComboBox* com a descrição *Privilegies For*. Nesse *ComboBox* podemos escolher as opções: *Roles, Users, Views, Triggers* e *Procedures*. São os objetos os quais podem receber privilégios. No botão *Salvar* podemos escolher como o IBExpert aplicará as mudanças. Se escolher *Save to Script Execute*, uma caixa de diálogo aparecerá com o *script* já pronto para execução, como no meu caso:

```
GRANT SELECT, INSERT, UPDATE, DELETE,
REFERENCES ON CLIENTES TO GRUPO_ALTERACAO;
```

Considerações finais

O nível de segurança em um banco de dados deve ser elevado de acordo com o sistema a ser desenvolvido. Há "n" fatores a se levar em consideração quando falamos em segurança da informação. Podemos citar milhares de casos onde a segurança não precisa ser tão elevada e outros tantos milhões que devem ser profundamente estudados. Cada caso deve ser avaliado individualmente no intuito de prover a melhor solução ao cliente.

Um fator importante que deve se levado em consideração é que o arquivo *security.fdb* que acompanha o banco de dados Firebird, é um arquivo separado. O fato do *security.fdb* ("arquivo que guarda os usuários e senhas do Firebird") ser separado do *SGBD* causa uma pequena falha na segurança no Firebird, já que uma pessoa mal intencionada poderia facilmente roubar o arquivo *.fdb* de nosso cliente, instalar o Firebird em qualquer máquina e abrir o banco roubado usando a senha SYSDBA/masterkey.

A nova vesão do Firebird, a 3.0, batizada de Vulcan vem sendo aprimorada em uma série de fatores e um deles diz respeito a essa "falha". O gerenciamento de usuário vem sendo embutido no banco, ou seja, a partir da versão 3.0 o *security.fdb* será interno. Em outras palavras, o SGBD de seu cliente poderá ter sua própria senha *SYSDBA* impossibilitando a abertura do mesmo sem conhecimento da senha. Enfim, segurança nunca é demais.

Conclusão

Nesse artigo vimos algumas dicas para melhorar a segurança em nossos aplicativos quando se trata de banco de dados. Uma boa dica é estudar com bastante dedicação e criar seus próprios mecanismos de segurança envitando que seus usuários burlem o sistema ou mesmo cometam erros. Até a próxima em um forte abraço. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

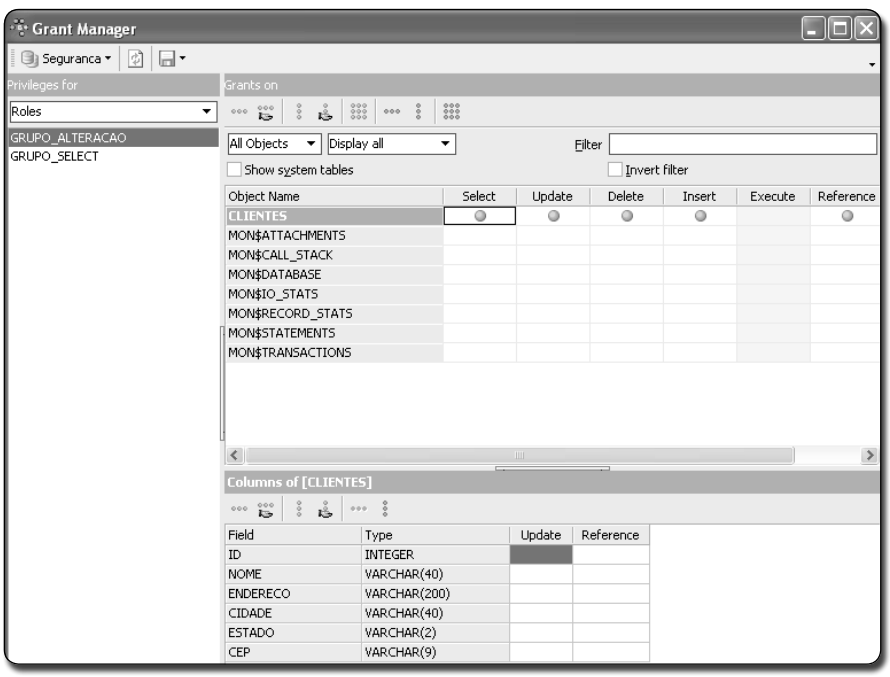
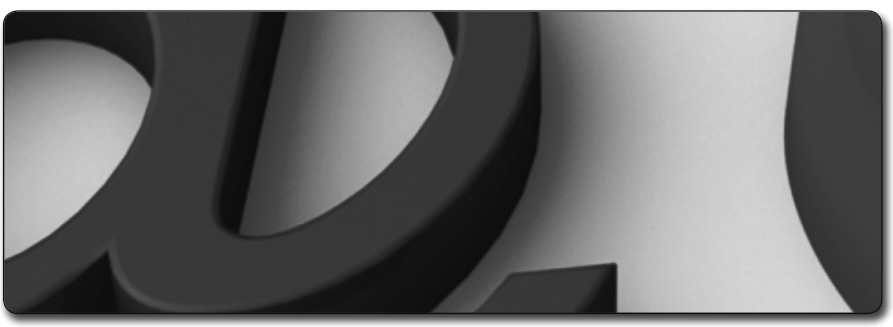


Figura 8. Grant e Revoke diretamente no IBExpert



Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Desenvolvendo Web Sites com PHP e Firebird

Veja como utilizar o banco de dados Firebird em aplicações PHP

Resumo DevMan

O banco de dados MySQL e a linguagem PHP se mostraram uma dupla infalível no que diz respeito a linguagem versus banco de dados. Sem dúvida nenhuma é a dupla que deu mais certo na internet até hoje. O MySQL é considerado o banco de dados mais rápido entre os SGDB's freeware disponíveis na rede mundial. Em conjunto com o PHP torna-se uma excelente ferramenta. Infelizmente o MySQL tornou-se um banco comercial e por esse motivo muitos desenvolvedores tem procurado alternativas para substituí-lo.

Desde as suas primeiras versões, o PHP possui suporte ao Interbase/Firebird. Na verdade ao Interbase, mas como os bancos são bem parecidos, podemos facilmente conectar uma página PHP a um SGBD Firebird.

É isso que veremos nesse artigo. Aprenderemos a desenvolver uma aplicação simples com acesso ao banco Firebird e de quebra ainda veremos como enviar e-mails.

Nesse artigo veremos

- Criação de um site simples totalmente feito em PHP;
- Inclusão de cadastros em banco de dados Firebird;
- Envio de e-mails.

Qual a finalidade

- Com esse artigo entenderemos os principais aspectos da linguagem PHP no desenvolvimento de aplicações para Web.

Quais situações utilizam esses recursos?

- Veremos nesse artigo uma extensão dos artigos anteriores, onde vimos as principais características da linguagem. Podemos criar qualquer tipo de aplicação on-line utilizando os recursos aqui mostrados.

Adriano Santos
falecom@adrianosantos.pro.br
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Assim como nosso mascote *DevMan* mencionou, o PHP é muito bem visto em conjunto com o MySQL, seu parceiro das antigas. O PHP foi criado em 1994 por *Ramus Lerdorf* como um CGI escrito em C, permitindo um número limitado de comandos. Foi inicialmente foi chamado de *Personal Home Page Tools*. *Lerdorf* criou a linguagem para processar formulários ao qual deu o nome de FI("Form Interpreter"). A junção de ambas linguagens originou na primeira versão compacta da linguagem: PHP/FI.

Em 1997 aconteceu uma grande contribuição à linguagem, quando o analisador sintático começou a ser programado. Diversas funcionalidades foram acrescidas à linguagem tais como suporte a

novos protocolos de Internet e também o suporte a grande maioria dos bancos de dados comerciais.

Todas essas melhorias resultaram na versão 3.0 da linguagem. A sigla PHP significa *PHP Hypertext Processor*, isso mesmo. O primeiro "P" é o próprio PHP o que soa um tanto estranho.

Praticamente junto com o PHP surgiu o MySQL. A primeira idéia do banco, digamos que se iniciou por volta de 1979 por uma equipe que tinha como principal objetivo: ter acesso às tabelas criadas na linguagem SQL. Em 1994, *Michael Widenius*, na companhia suíça TcX, criou o MySQL que por sua vez começou a crescer em popularidade por volta de 1995 com o lançamento de sua primeira versão. De lá pra cá tem crescido até a sua venda para a Sun Microsystems. Até então a dupla MySQL e PHP estavam sempre juntas.

O Firebird tornou-se, com toda certeza, um dos SGBD's *free* mais utilizados entre a comunidade Delphi, mas e na Web? O que será dele?

Nesse artigo veremos como o Firebird se comporta na Web dando apoio a uma das linguagens mais utilizadas para o desenvolvimento de Web Sites.

Faremos a instalação do PHP 5.2.6 no Windows e em conjunto com o IIS ("Internet Information Server"). Desenvolveremos uma aplicação capaz de cadastrar e listar usuários em um banco Firebird.

Nas edições de 89 a 92 vimos os principais aspectos da linguagem. Vimos também condicionais, laços, *arrays*, enfim, todos os detalhes sobre PHP. Agora vamos desenvolver uma aplicação com PHP puro, prepare-se.

Baixando e instalando o PHP 5.2.6

A primeira providência a se tomar para desenvolver nossa aplicação teste é baixar e instalar o PHP 5.2.6 em nosso micro. A versão 5 está muito mais fácil de instalar do que as verões anteriores. Agora é possível fazer a instalação com apenas alguns cliques. Para isso, abra o seu *browser* preferido e acesse o site oficial da linguagem em *www.php.net*. Em seguida acesso o link *downloads*. Na página de *downloads* encontramos

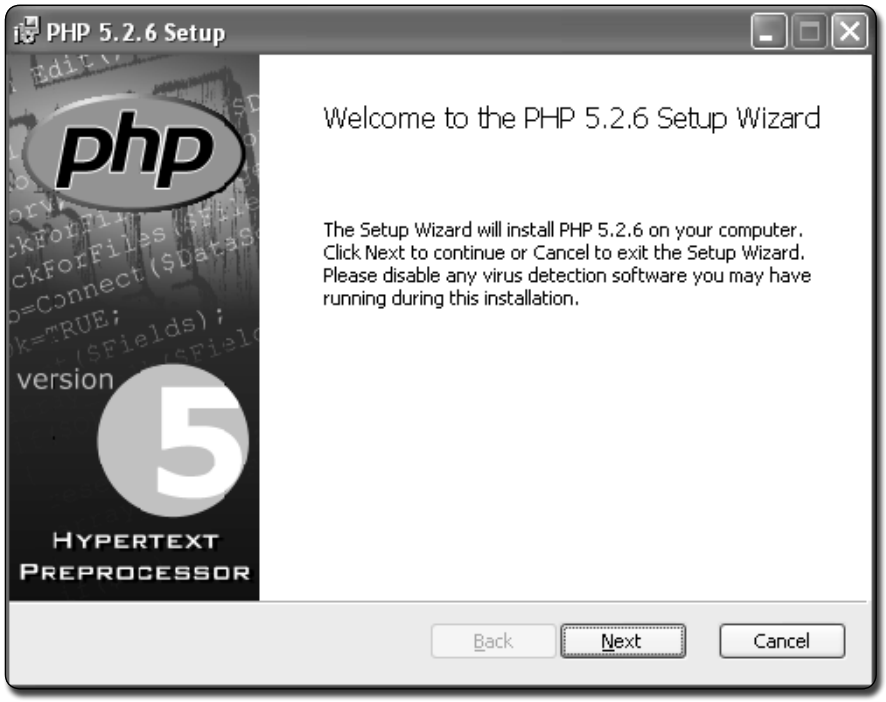


Figura 1. Instalação do PHP 5.2.6

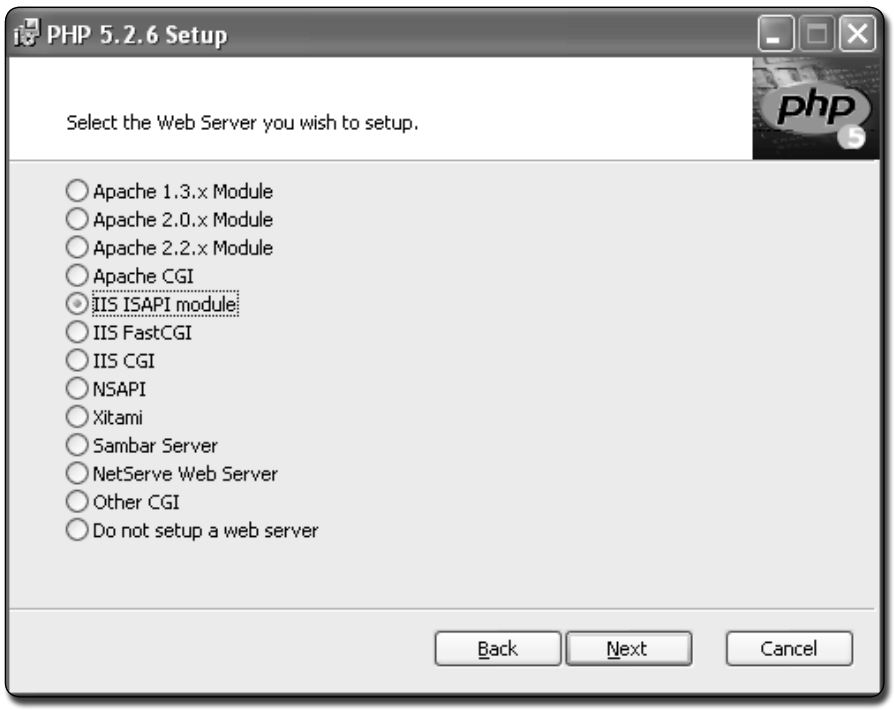


Figura 2. Escolhendo o modo de instalação do PHP

diversos *links* e versões do PHP. Faça o *download* do *PHP 5.2.6 installer* em *Windows Binaries* ou, se preferir, digite na barra de endereços o link *www.php.net/get/php-5.2.6-win32-installer.msi/from/a/mirror*. Você será redirecionado para uma página com diversos *mirrors*("servidores de download") e poderá escolher um deles para baixar.

Após baixado o arquivo em sua máquina, clique duas vezes sobre *php-5.2.6-win32-installer.msi* e vamos à instalação (**Figura 1**). Clique em *Next* na primeira tela e em seguida escolheremos o modo que o PHP será instalado (**Figura 2**). Escolha *IIS ISAPI module*. Como pode ver, podemos instalar o PHP de diversas formas e em diversos servidores de Web, incluindo o *Apache*.

Basicamente, não há mais nada de extraordinário na instalação do PHP. Basta seguir o restante do *Wizard* que termos a linguagem instalada e rodando em nosso Windows. O que devemos fazer agora é informar ao IIS que o PHP está presente e por isso precisamos configurá-lo.

Assim que a instalação terminar, entre no *Painel de Controle* do *Windows* e acesse *Ferramentas Admnistrativas>Internet Information Services*. Clique com o botão direito do mouse sobre o item *Default Web Site* ("Sites da Web Padrão") localizado à esquerda da janela. Expanda a árvore de pastas se não conseguir encontrar o item mencionado. Agora clique em *Filtros ISAPI* (**Figura 3**) e em seguida no botão *Adicionar*. Digite o nome do filtro e em seguida clique em *Procurar*.

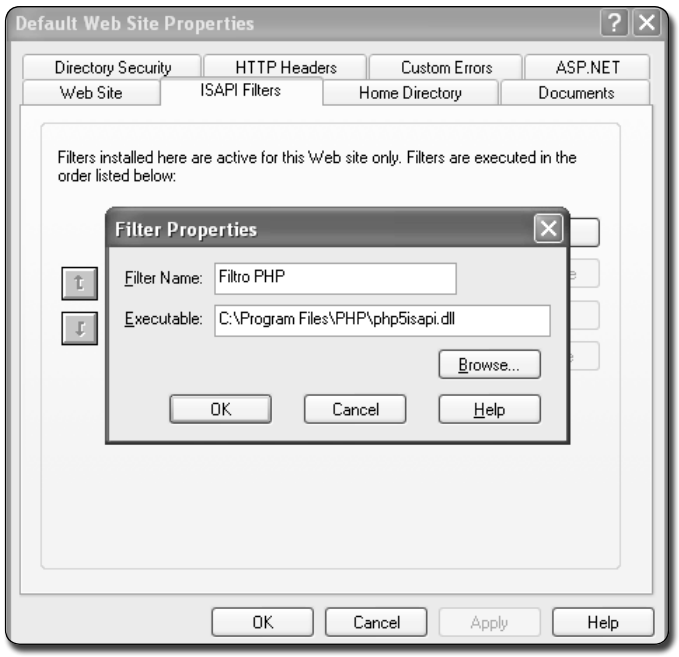


Figura 3. Adicionando o filtro PHP

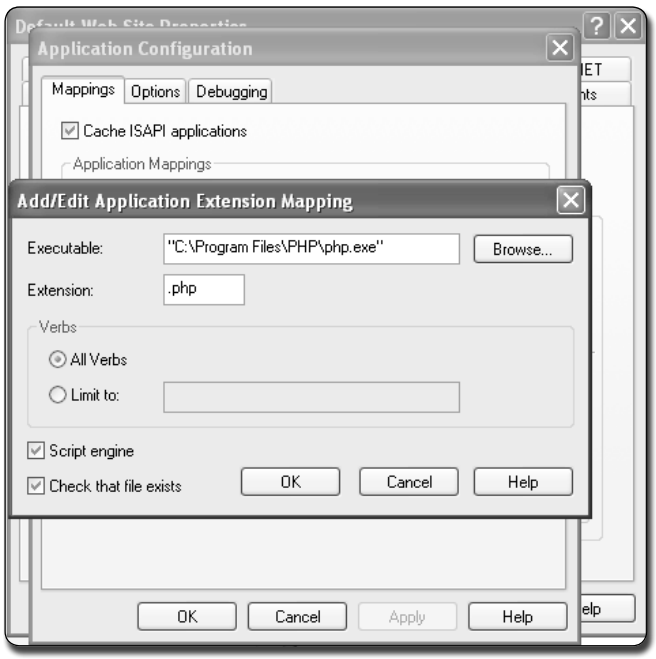


Figura 4. Adicionando a extensão do PHP

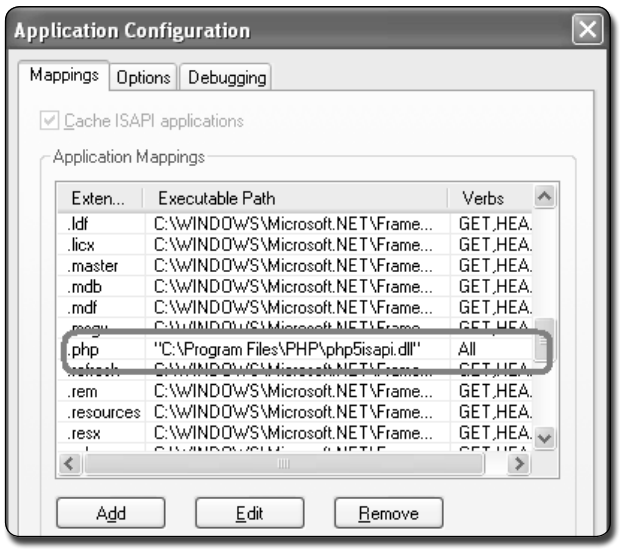


Figura 5. Extensão adicionada

Figura 6. Uso do PHP no Windows

Localize e escolha o arquivo *php5isapi.dll* presente no diretório *C:\Arquivos de Programas\PHP* caso não tenha mudado o caminho durante a instalação.

Agora vá até a aba *Diretório Home* e clique em *Configuration*. Nesse ponto adicionaremos a extensão *.php* para que o IIS reconheça-a. Clique em *Procurar* e localize o arquivo *php.exe* no diretório de instalação da linguagem(**Figura 4**). Na **Figura 5** você encontra um exemplo de como ficará sua tela.

Por fim clique em *Documentos* e em seguida em *Adicionar*. Agora faremos a adição do arquivo *index.php*. Ele será o arquivo padrão procurado pelo servidor PHP sempre que uma página for requisitada. Digite *index.php* e confirme.

O último passo é criarmos um diretório virtual no IIS para receber os arquivos de nosso web site. Para isso dê um clique de direita no IIS e escolha *Novo>Diretório Virtual* no menu *pop-up*. Clique em *Próximo* na primeira tela e na tela seguinte digite o *alias*("apelido") para a página e continue.

Marque a opção *Leitura* (**Figura 6**) se não estiver marcada. Continue com *Próximo* e, prosseguindo, informe o caminho para os arquivos do site, em nosso caso guardaremos em *C:\Teste-PHP*. Por fim clique em *Finish*. Todos os arquivos de nosso web site serão salvos nessa pasta de agora em diante. Com isso finalizamos a instalação.

> **Nota**: Em algumas situações, é necessário copiar o arquivo *php.ini* do diretório de instalação do PHP para o diretório *System32* do Windows.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Campo	Tipo/Tamanho
ID	Integer Not Null Autoinc
NOME	Varchar(50)
EMAIL	Varchar(50)
APELIDO	Varchar(10)

Tabela 1. Tabela NewsLetter

Criando o banco de dados

Um passo importante para o nosso exemplo certamente é criar o banco de dados a ser utilizado. Não entrarei em detalhes quanto à instalação do Firebird, apenas a criação do banco.

Para iniciar o projeto, vamos primeiramente criar a base de dados e a tabela necessária para a tela de cadastro. A tabela que criaremos terá o nome de "Newsletter". Faremos uma aplicação onde o internauta poderá se cadastrar para receber futuras notícias. Neste artigo, criaremos o banco de dados utilizando a ferramenta IBExpert em sua versão *Standard*. Por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de *download* da ferramenta. Após isso, abra o IBExpert e vamos criar o banco de dados usando a opção *Script Executive* presente no menu *Tools*.

Com a tela de *scripts* aberta, digite o código da **Listagem 1**. Nele estão

Listagem 1. Criação do banco de dados

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
CREATE DATABASE '<Caminho>NomeDoBanco.gdb'
USER 'SYSDBA'
PAGE_SIZE 8091
DEFAULT CHARACTER SET WIN1252;

CREATE GENERATOR GEN_NEWSLETTER_ID;
SET GENERATOR GEN_NEWSLETTER_ID TO 2;

CREATE TABLE NEWSLETTER (
    ID       INTEGER NOT NULL,
    NOME     VARCHAR(50),
    EMAIL    VARCHAR(50),
    APELIDO  VARCHAR(10)
);

INSERT INTO NEWSLETTER (ID, NOME, EMAIL, APELIDO)
  VALUES (1, 'ADRIANO', 'falecom@adrianosantos.pro.br',
  'ADRIANO');
INSERT INTO NEWSLETTER (ID, NOME, EMAIL, APELIDO)
  VALUES (2, 'GUINTHER', 'guinther@devmedia.com.br',
  'GUINTHER');

ALTER TABLE NEWSLETTER ADD CONSTRAINT PK_NEWSLETTER
  PRIMARY KEY (ID);

SET TERM ^ ;
/* Trigger: NEWSLETTER_BI */
CREATE TRIGGER NEWSLETTER_BI FOR NEWSLETTER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = GEN_ID(GEN_NEWSLETTER_ID,1);
END
^
SET TERM ; ^
```

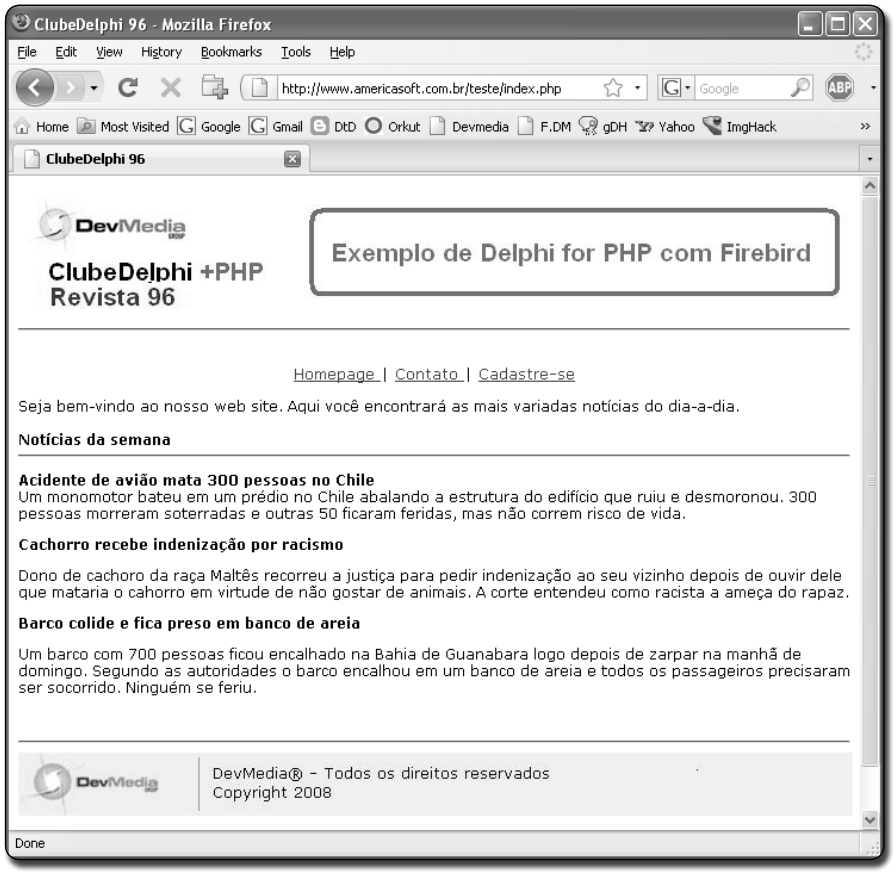


Figura 7. Exemplo de Web Site

Listagem 2. Código da página cabecalho.php

```
<html>
  <head>
    <title>ClubeDelphi 96</title>
  </head>
  <body>
    <p>
      <img src="cabecalho.jpg" width="700"
        height="96"></p>
    <hr>
    <br>
  </body>
</html>
```

Listagem 3. Código da página rodape.php

```
<html>
  <head>
    <title>ClubeDelphi 96</title>
  </head>
  <body>
    <p>
      <img src="rodape.jpg" width="700"
        height="96"></p>
    <hr>
    <br>
  </body>
</html>
```

Listagem 4. Código CSS do arquivo estilo.css

```
.paragrafo{
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 12px;
}
.titulo{
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 12px;
font-weight: bold}
```

contidos os esquemas para a criação do banco de dados assim como de cada tabela do sistema. Digitado o *script*, basta executá-lo usando o botão *Run Script* ou pressione a tecla F9. A estrutura completa da tabela *NewsLetter* encontra-se na **Tabela 1**.

> **Nota:** Substitua *<Caminho>\NEWS.FDB* pelo diretório do banco de dados e seu nome, ex: *C:\BancoDeDados\News.fdb*.

Como pode notar, a tabela é bem simples. O campo *ID* é do tipo inteiro e prevê a criação de um *Generator* para auto-numeração. Poderíamos adicionar uma série de outros campos, mas para nosso exemplo não é necessário.

Criando a aplicação

Para criar nosso sistema vamos fazer uso do bloco de notas para desenvolver nossas telas. Usaremos HTML e PHP puro para desenhar e fazer as devidas programações. Por isso abra uma janela do bloco de notas e vamos iniciar o nosso sistema. Nossa tela principal terá a aparência da **Figura 7.**

Note que estamos usando um cabeçalho e um rodapé. Na parte superior ainda temos um menu que é padrão para todo Web Site. Eu preferi criar uma imagem no *Paint* mesmo para poder exemplificar a criação desse cabeçalho e rodapé, portanto crie duas novas imagens no programa que desejar e salve-as como "cabecalho.jpg" e "rodape.php" no diretório da nossa aplicação, respectivamente. Os dois primeiros arquivos que criaremos são "cabecalho.php" e "rodape.php".

O código é bem simples, basta usarmos a *tag* HTML *<img>* e inserir a imagem. Veja como é simples criar o código para isso na **Listagem 2**. Logo de início estamos declarando a *tag* HTML, responsável por indicar que o documento é uma página HTML. Dentro da *tag <head>*, determinamos tudo que é comum para o cabeçalho do Web Site, nesse caso o título envolvido pelas *tags <title>* e *</title>*. Logo em seguida vemos um *body* da página, ou seja, o corpo dela. Estamos, nesse ponto, inserindo uma imagem através da linha:

```
<img src="cabecalho.jpg"
     width="700"
     height="96">
</p>
```

Feito isso, toda vez que uma página for carregada esse cabeçalho também será, ou seja, teremos uma imagem de cabeçalho padrão. Salve o arquivo como "cabecalho.php" assim como mencionado.

Na **Listagem 3** vemos o código do *rodape.php* que por sinal é idêntico, tendo como alteração apenas a imagem. Veja que precisamos ter um certo conhecimento de HTML para se programar em PHP puro, já que mesclamos as duas linguagens em uma só. Mais adiante veremos como incluir código PHP em HTML e tornar nossas páginas dinâmicas.

O próximo passo será criamos um arquivo de estilo em CSS ("Cascading Style Sheets") para melhoramos a aparência de nossas páginas. O CSS nada mais é que uma página de estilos. Podemos digitá-los diretamente em uma nova janela do bloco de notas e salva-lo com o nome que desejarmos. Normalmente usamos a extensão *.css*, mas não é uma regra, portanto podemos colocar a extensão que quisermos.

Abra uma nova janela do bloco de notas e digite o texto da **Listagem 4**. Não entrarei em detalhes sobre o CSS por não fazer parte do escopo desse artigo, mas de uma maneira mais resumida vejamos o que o código está fazendo. Estamos declarando duas classes que são: *.paragrafo* e *.titulo*. Elas serão usadas para formatar automaticamente e dentro do padrão, todos os textos de nossas páginas. A classe *.paragrafo* está sendo configurada para exibir o texto na fonte *Verdana* e tamanho *12px*. Já em *.titulo* repetimos os primeiros atributos e acrescentamos o atributo de negrito. Para entendermos melhor, visualize as **Figuras 7** e **8**, sendo que a **Figura 8** não possui vínculo com o arquivo CSS.

Voltando ao nosso exemplo, salve o arquivo criado anteriormente na pasta de nosso web site com o nome "estilo.css".

Ainda nas figuras que acabamos de ver, note que temos um menu com os itens *Homepage*, *Contato* e *Cadastre-se*. Faremos esse menu em um arquivo separado, assim podemos utilizar o método *include()* do PHP para incluir em todas as páginas nosso menu juntamente com o cabeçalho

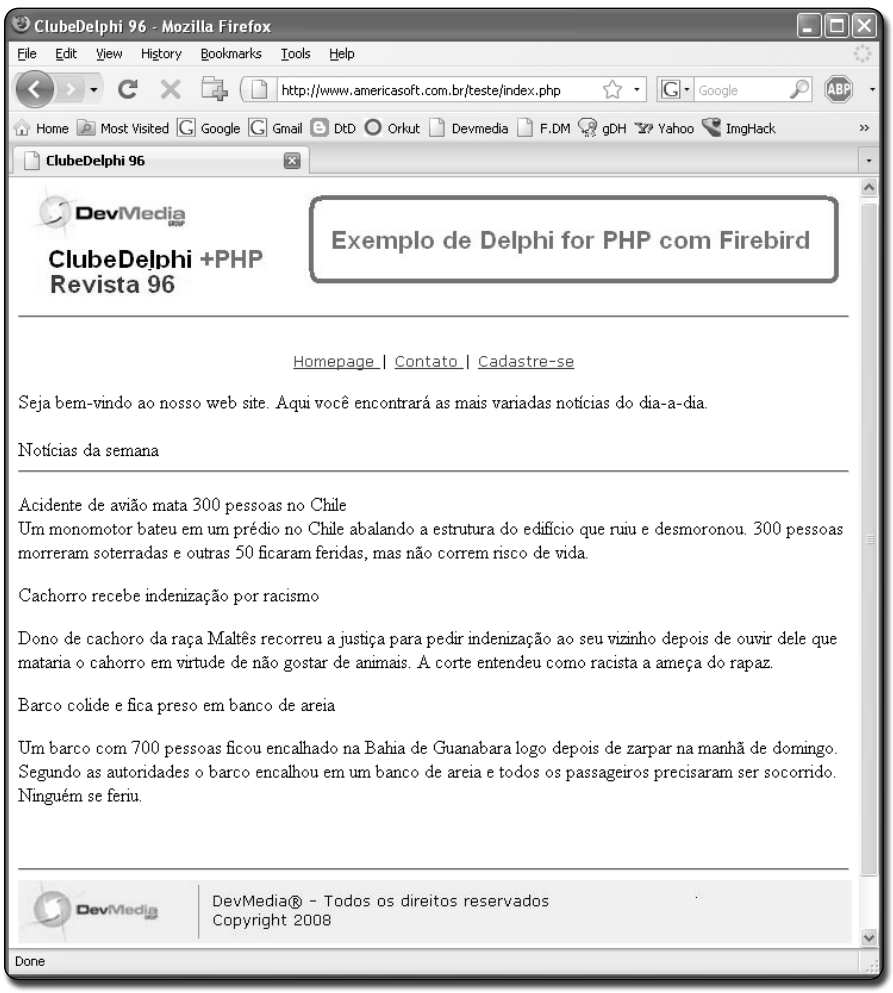


Figura 8. Página sem estilos CSS

Listagem 5. Criação do Menu da aplicação

```
<div align="center">
  <font face="Verdana, Arial, Helvetica, sans-serif" size="2">
    <a href="index.php"> Homepage </a> |
    <a href="contato.php">Contato </a> |
    <a href="cadastrar.php?Acao=C">Cadastre-se</a>
  </font>
</div>
```

Nota do DevMan

Cascading Style Sheets, ou simplesmente CSS, é uma linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML. Seu principal benefício é prover a separação entre o formato e o conteúdo de um documento.

Ao invés de colocar a formatação dentro do documento, o desenvolvedor cria um link (ligação) para uma página que contém os estilos, procedendo de forma idêntica para todas as páginas de um portal. Quando quiser alterar a aparência do portal basta, portanto, modificar apenas um arquivo.

Com a variação de atualizações dos navegadores (browsers) como Internet Explorer que ficou sem nova versão de 2001 a 2006, o suporte ao CSS pode variar. O Internet Explorer 6, por exemplo, tem suporte total a CSS1 e praticamente nulo a CSS2. Navegadores mais modernos como Opera, Internet Explorer 7 e Mozilla Firefox tem suporte maior, inclusive até a CSS 3, ainda em desenvolvimento.

A interpretação dos browsers pode ser avaliada com o teste Acid2, que se tornou uma forma base de revelar quão eficiente é o suporte de CSS, fazendo com que a nova versão em desenvolvimento do Firefox seja totalmente compatível a ele assim como o Opera já é.

Eis um exemplo de CSS:

```
body
{
  font-family: Arial, Verdana, sans-serif;
  background-color: #FFF;
  margin: 5px 10px;
}
```

e rodapé. Abra uma nova janela ou crie um novo arquivo no Bloco de Notas e digite o código da **Listagem 5**. Nesse código estamos incluindo três links na página para simularmos o menu. Mais uma vez vemos um código HTML, ainda mais simples. No código estamos os valendo da *tag <href>* para criar links. A *tag <font>* é usada para formatar a fonte do menu. Veja que teremos três arquivos em nossa aplicação: *index.php*, *contato.php* e *cadastrar.php*. O primeiro deles é efetivamente nossa página default, ou seja, a primeira a ser carregada ao digitarmos o endereço do site. A página *contato.php* é que usaremos para enviar mensagens, nas verdade quem usará a tela de envio será o usuário final. Ele poderá entrar em contato conosco a hora que quiser apenas entrando no site e nos enviando um e-mail. Por fim, o arquivo *cadastrar.php* será usado para cadastrar-se em nossa *newsletter*. Isso significa que o internauta que desejar se cadastrar em nosso site poderá fazer isso automaticamente através do link *Cadastrar-se*. Dessa forma os dados dele já ficarão disponíveis diretamente na base de dados. E é aí que entra o Firebird.

Criando a página Index.php

Nesse momento já podemos começar a criar as principais páginas de nosso site. Agora que temos os arquivos *cabecalho.php*, *rodapé.php*, *estilo.css* e *menu.php*, abra uma nova janela do Bloco de Notas e vamos digitar o código da página principal, *index.php*. Digite o código da **Listagem 6** e vamos ao detalhamento do arquivo. Para quem é leigo fica difícil descobrir as diferenças, mas vamos lá. A primeira delas está no fato de termos código PHP e código HTML misturados no texto do arquivo. Logo de início encontramos o trecho de fonte a seguir:

```
<?php
  include('cabecalho.php');
?>
```

Esse código é PHP e pode ser identificado por dois motivos.

1. O sinal de "?" no início da *tag*. Sempre que iniciamos um código PHP usamos a interrogação precedida do sinal menor para abrir a linguagem PHP. Em seguida, ao término da codificação, usamos o contrário, ou seja, sinal de interrogação mais sinal de maior;
2. A palavra *php* logo no início. Não é obrigatório, mas fica mais visível no código-fonte quando ela está presente.

> **Nota:** É possível usarmos as *tags* <% e %> do antigo ASP para abrirmos e fecharmos codificações PHP em um arquivo *.php*. Porém, para isso precisamos configurar a variável *asp_tags* para *on* ex.: *asp_tags = on*;

Dentro das *tags* de PHP estamos chamando o método i*nclude()* do PHP indicando pra ele o arquivo *cabecalho.php*. Isso fará com que esse arquivo seja incluído na página exatamente onde foi digitado, ou seja, no topo do site. Essa técnica é bastante interessante, pois a qualquer momento podemos modificar a Figura ou qualquer coisa na parte superior que todo o site estará modificado.

Outra grande mudança está localizada dentro da *tag <head>*. Nessa tag fizemos o link direto com o arquivo *estilo.css* para que a página passe a enxergar os estilos do site. O código que faz isso é visto a seguir:

```
<link href="estilo.css"
      rel="stylesheet"
      type="text/css">
```

Chegamos o corpo do *index.php*, onde faremos a impressão do que será visualizado pelo usuário final. Note que a primeira linha do *body* é novamente um código PHP. Estamos fazendo um *include()* do arquivo *menu.php*, para que possamos ter acesso a todas as páginas do site.

```
<? include('menu.php'); ?>
```

Listagem 6. Código da página index.php

```
<?php
  include('cabecalho.php');
?>

<html>
<head>
  <title>ClubeDelphi 96</title>
  <link href="estilo.css" rel="stylesheet"
    type="text/css">
</head>
<body>
  <? include('menu.php'); ?>

  <p class="paragrafo">
    Seja bem-vindo ao nosso web site. Aqui você
    encontrará as mais variadas notícias do
    dia-a-dia.<br>
  <br>
  <span class="titulo">
    Not&iacute;cias da semana<br><hr>
  </span></p>
  <span class="titulo">Acidente de avi&atilde;o
    mata 300 pessoas no Chile
  </span><br>
  <span class="paragrafo">Um monomotor bateu em um
    pr&eacute;dio no Chile abalando a estrutura do
    edif&iacute;cio que ruiu e desmoronou. 300 pessoas
    morreram soterradas e outras 50 ficaram feridas,
    mas n&atilde;o correm risco de vida.
  </span>
  <p class="titulo">Cachorro recebe
    indeniza&ccedil;&atilde;o por racismo
  </p>
  <p class="paragrafo">Dono de cachoro da
    ra&ccedil;a Malt&ecirc;s recorreu a justi&ccedil;a
    para pedir indeniza&ccedil;&atilde;o ao seu vizinho
    depois de ouvir dele que mataria o cahorro em
    virtude de n&atilde;o gostar de animais. A corte
    entendeu como racista a ame&ccedil;a do rapaz.
  </p>
  <p class="titulo">Barco colide e fica preso em
    banco de areia
  </p>
  <p class="paragrafo">Um barco com 700 pessoas
    ficou encalhado na Bahia de Guanabara logo
    depois de zarpar na manh&atilde; de domingo.
    Segundo as autoridades o barco encalhou em
    um banco de areia e todos os passageiros
    precisaram ser socorrido. Ningu&eacute;m se feriu.
  </p>
  <p class="paragrafo">
     
  </p>
</body>
</html>
<? include('rodape.php'); ?>
```

Em seguida iniciamos a impressão das notícias, que em nosso exemplo estão fixas, mas poderiam vir de um banco de dados só de notícias. A parte mais interessante aqui é que todas as notícias estão envolvidas pela *tag <p span class>* e que no *class* estamos informando a classe *.paragrafo* ou *.titulo*. Isso faz com que as classes determinadas no arquivo *estilo.css* sejam aplicadas.

Por fim fechamos as *tags HTML*, *Body* e a inclusão do arquivo *rodapé.php* através do método *include()*. Salve o arquivo como "index.php", sempre na pasta que criamos no início do artigo. Nesse ponto podemos testar as páginas, portanto salve o *index.php*, abra o seu *browser* preferido e digite na barra de endereços *http://localhost/pasta_virtual_do_site/index.php*.

Enviando e-mail com PHP

O envio de e-mails com PHP é muito simples. Ao contrário de outras linguagens de programação para Web, no PHP apenas precisamos invocar o método *mail()* e passar os parâmetros para ele. O resto é feito pela função e facilmente a mensagem será enviada.

Crie um novo arquivo e digite o código da **Listagem 7**. Salve o arquivo como "contato.php". A grande sacada desse código está no fato de que montamos tudo em modo texto e o resultado será o apresentado na **Figura 9**. Veja que é simples do design, porém funcional. Estamos incluindo uma *tag form* com seu *Action* apontado para *enviar.php* conforme vemos a seguir:

```
<form name="frmEnviar"
      method="post"
      action="enviar.php">
```

Essa tag em HTML monta o formulário que acabamos de ver na **Figura 9**. O atributo *action* é a ação que queremos que seja disparada. Nesse caso estamos desviando para o arquivo *enviar.php*, que por sua vez fará o envio da mensagem.

No arquivo *enviar.php* capturaremos os dados que estarão chegando e enviaremos por e-mail. Veja o código do arquivo *enviar.php* na **Listagem 8**. Digite-o como na **Listagem 8** e salve-o como "enviar.php". O código é simples. Primeiro salvamos em uma variável o e-mail para quem vai a mensagem, nesse caso para *falecom@*

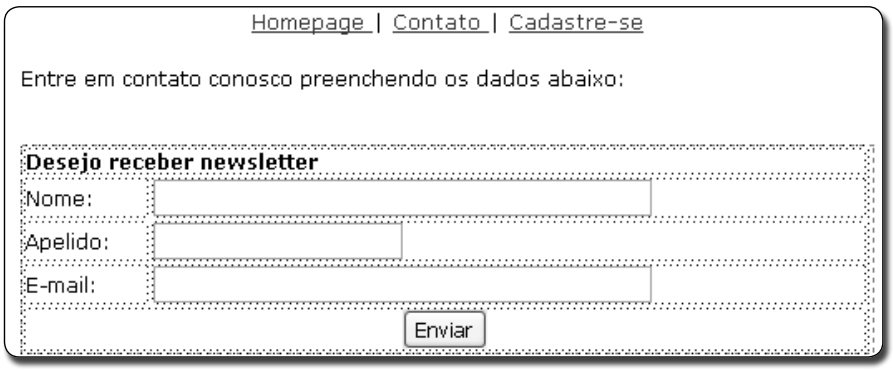


Figura 9. Exemplo de tela de contato

Listagem 7. Código da página contato.php

```
<?php
  include('cabecalho.php');
?>
<html>
<head>
  <title>ClubeDelphi 96</title>
  <link href="estilo.css" rel="stylesheet"
    type="text/css"></head>
</head>
<body>
<? include('menu.php'); ?>
<p class="paragrafo">
  Entre em contato conosco preenchendo os
  dados abaixo:<br>
</p>
<form name="frmEnviar" method="post"
  action="enviar.php">
  <table width="511" border="0">
    <tr>
      <td colspan="2" class="titulo">
        Desejo receber newsletter
      </td>
    </tr>
    <tr>
      <td class="paragrafo">
        Nome:
      </td>
      <td>
        <input name="txtNome" type="text" id="txtNome" size="50" maxlength="50">
      </td>
    </tr>
    <tr>
      <td class="paragrafo">
        Apelido:
      </td>
      <td>
        <input name="txtApelido" type="text" id="txtApelido" size="25" maxlength="10">
      </td>
    </tr>
    <tr>
      <td class="paragrafo">
        E-mail:
      </td>
      <td>
        <input name="txtEmail" type="text"
          id="txtEmail" size="50" maxlength="50">
       </td>
    </tr>
    <tr>
      <td colspan="2">
        <div align="center">
          <input name="btnEnviar" type="submit"
            id="btnEnviar" value="Enviar">
        </div>
      </td>
    </tr>
  </table>
</form>
<p class="paragrafo">
<br>
</p>
</body>
</html>
<?php
  include('rodape.php');
?>
<?php
include('rodape.php');
?>
```

Listagem 8. Código de gravação dos dados no banco

```
<?php
include('cabecalho.php');
?>
<html>
  <head>
    <title>ClubeDelphi 96</title>
    <link href="estilo.css" rel="stylesheet"
      type="text/css">
  </head>
<body>
<? include('menu.php'); ?>

<? if ($Acao=="C"){ ?>

<p class="paragrafo">
  Preencha os dados abaixo e clique em gravar:<br>
</p>
<form name="frmEnviar" method="post"
  action="cadastrar.php?Acao=I">
  <table width="511" border="0">
    <tr>
      <td colspan="2" class="titulo">Desejo receber newsletter</td>
    </tr>
    <tr>
      <td class="paragrafo">Nome:</td>
      <td>
        <input name="txtNome" type="text" id="txtNome" size="50" maxlength="50">
      </td>
    </tr>
    <tr>
      <td class="paragrafo">Apelido:</td>
      <td>
        <input name="txtApelido" type="text" id="txtApelido" size="25" maxlength="10">
      </td>
    </tr>
    <tr>
      <td class="paragrafo">
       E-mail:
     </td>
     <td>
       <input name="txtEmail" type="text" id="txtEmail" size="50" maxlength="50">
     </td>
    </tr>
    <tr>
      <td colspan="2"><div align="center">
        <input name="btnGravar" type="submit" id="btnGravar" value="Gravar">
      </div></td>
    </tr>
  </table>
</form>
<?
}else{
  $ID = 0;
  $NOME= StrToUpper($HTTP_POST_VARS["txtNome"]);
  $EMAIL= StrToUpper($HTTP_POST_VARS["txtEmail"]);
  $APELIDO= StrToUpper($HTTP_POST_VARS["txtApelido"]);

  $conexao = ibase_connect('<caminho>/newsletter.gdb',
    'usuario', 'senha') or die ("Erro na conexão");
  $max_id = ibase_query("SELECT MAX(ID) FROM NEWSLETTER");
  $max = ibase_fetch_row($max_id);
  $ID = $max[0] + 1;
  $grava = ibase_query("INSERT INTO NEWSLETTER
    VALUES('$ID', '$NOME', '$EMAIL', '$APELIDO')");

  $para = $EMAIL;
  $assunto  = "Cadastro efetuado";
  $mensagem = "$NOME, você está recebendo este
    e-mail porque cadastrou-se em nossa newsletter";

  $headers = "Content-Type: text/plain;
    charset=\"iso-8859-1\"\r\n";
  $headers .= "From: 'falecom@adrianosantos.pro.br'
    <falecom@adrianosantos.pro.br>";

  mail($para, $assunto, $mensagem, $headers);

  echo "<br><br><br><br>";
  echo "<span class='paragrafo'>Parabéns, seus dados
    foram cadastrados com sucesso</span>";
  echo "<br><br>";
  echo "<span class='paragrafo'>Você receberá um
    e-mail com a confirmação de seus dados.</span>";
  echo "<br><br><br><br>";
}
?>
</body>
</html>
<?php
include('rodape.php');
?>
```

adrianosantos.pro.br. Em seguida usamos as variáveis globais *$HTTP_POST_VARS* para receber os valores passados pela tela anterior. Por fim enviamos o e-mail usando a função *mail()*.

Gravando no banco de dados

Sem dúvida nenhuma a parte mais interessante do artigo: a gravação dos dados. Faremos agora um pequeno formulário para gravarmos *Nome*, *E-mail* e *Apelido* do internauta no banco de dados Firebird.

Vamos criar um novo arquivo no bloco de notas com o nome "cadastrar.php". Nele digitaremos o código da **Listagem 8**. A explicação para o código fonte talvez seja longa, mas bem interessante. Vejamos então.

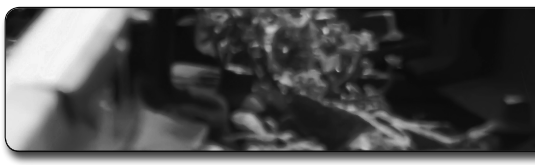
Do início da página até a primeira linha do *body* da página já sabemos exatamente o que o sistema está fazendo, mas se notarmos bem veremos que já um elemento diferente na página. Veja o *if* que estamos fazendo na segunda linha do *body*:

```
<? if ($Acao=="C"){ ?>
```

Estamos dizendo que se o parâmetro *$Acao* for igual a *C* então faremos a impressão normal do cadastro. O parâmetro é passado por meio de um link ou digitação direta na barra de título. Voltemos ao menu. A linha que imprime o link para a tela de cadastro é:

```
<a href="cadastrar.php?Acao=C">Cadastre-se</a>
```

Veja que colocamos o nome da página seguida da *Acao*: "cadastrar.php?Acao=C". Toda vez que a página *cadastrar.php* perceber um *C* no parâmetro, então o formulário será exibido. Vamos analisar mais de perto isso tudo. Em PHP podemos carregar a mesma página e executar o código analisando os parâmetros que ela recebe, assim como qualquer outra linguagem ou mesmo o HTML. Algo ainda mais interessante estamos fazendo. Estamos criando um *if.. else* misturado com HTML. Na linha a seguir estamos iniciante a condicional e abrindo a chaves. Em seguida fechamos



a *tag* PHP. Isso significa que o *browser* vai executar o que vier depois e retornará ao código PHP assim que encontrar a finalização da chave. Observe:

```
<? if ($Acao=="C"){ ?>
  //Código HTML
<? }else{
  //Código PHP
}?>
```

No código anterior, a chave que fecha a condicional está logo depois dos comandos HTML e é finalizado com um *else*. Isso é fantástico, pois mesclamos facilmente código PHP com código HTML.

A segunda parte do script prevê a inclusão dos dados resgatados pelo formulário, dentro da base de dados. Primeiro conectamos ao banco usando o método *ibase_connect*, passando para ele o *host do banco*, *usuário* e *senha*. Em seguida pegamos o *Max* da tabela para verificar qual o próximo *ID*. Feito isso executamos um *INSERT INTO* no banco de dados e enviamos um e-mail de confirmação. Uma mensagem também é exibida na tela conforme a **Figura 10.**

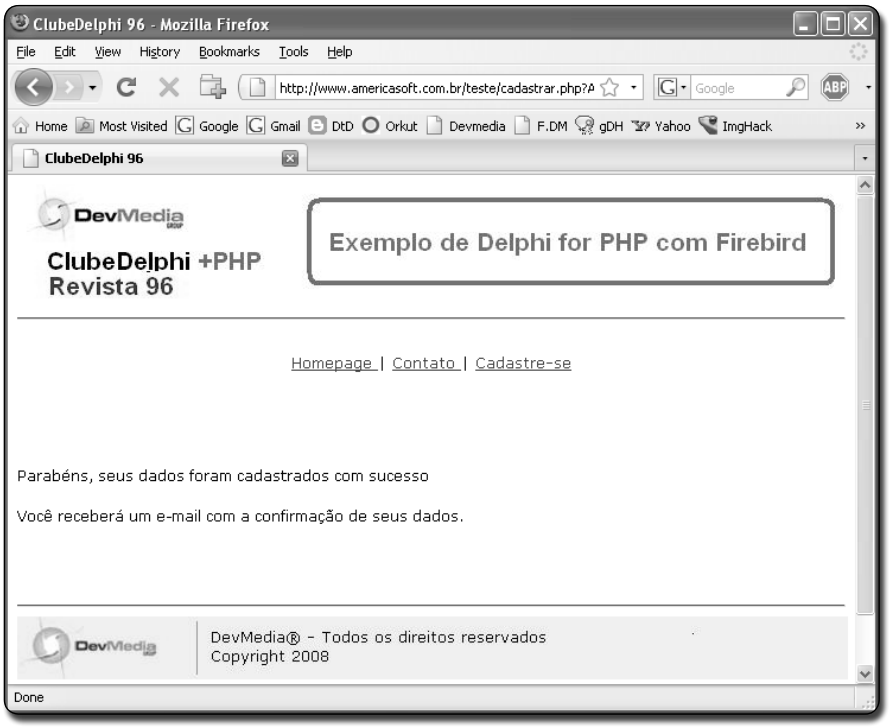


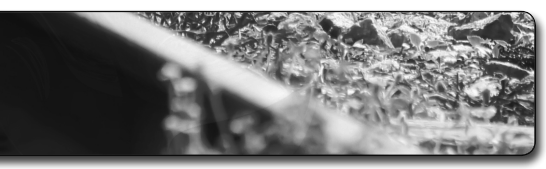
Figura 10. Confirmação dos dados cadastrados

Listando os usuários cadastrados

A última parte desse artigo consiste em listarmos os usuários cadastrados. Para isso criaremos uma tela de administrador chamada "admin.php", portanto crie novamente um arquivo no bloco de notas e salve-o com o nome sugerido. Dessa vez faremos a conexão com o banco de dados e listaremos todos os indivíduos cadastrados.

O código é bastante semelhante ao mostrado na seção anterior, porém faremos um laço *while* no resultado e imprimiremos em tela usando o método *print* do PHP (**Listagem 9**).

Como pode ver o código não é complexo. Estamos criando uma conexão com o servidor Firebird e em seguida executando uma consulta usando uma *select*. O método usado é o *ibase_query*, que retorna para a variável *$consulta* o resultado do filtro. Envolvemos todo o processo em um laço *while* que fará



Listagem 9. Listando usuários

```
<?php
include('cabecalho.php');
?>
<html>
  <head>
    <title>ClubeDelphi 96</title>
      <link href="estilo.css" rel="stylesheet"
        type="text/css">
  </head>
<body>
<? include('menu.php'); ?>
<p class="paragrafo">
  Visualize todos os usu&aacute;rios cadastrados
  em sua newsletter</p>
<table width="200" border="0">
  <tr>
    <td><span class="paragrafo">ID</span></td>
    <td><span class="paragrafo">Nome</span></td>
    <td><span class="paragrafo">Apelido</span></td>
   <td><span class="paragrafo">E-mail</span></td>
  </tr>
<?
  $conexao = ibase_connect('<caminho>/newsletter.gdb',
    'usuario', 'senha') or die ("Erro na conexão");
  $consulta = ibase_query("SELECT * FROM NEWSLETTER");

  $i = 0;
  while($resultados = ibase_fetch_row($consulta)){
    print "<tr>";
    print "  <td><span class='paragrafo'>" .
      $resultados[0] . "</span></td>";
    print "  <td><span class='paragrafo'>" .
      $resultados[1] . "</span></td>";
    print "  <td><span class='paragrafo'>" .
      $resultados[2] . "</span></td>";
    print "<td><span class='paragrafo'>" .
      $resultados[3] . "</span></td>";
    print "</tr>";
    $i++;
  }
?>
</table>
</body>
</html>
<?php
  include('rodape.php');
?>
```

a impressão enquanto a variável *$resultados* contiver valor. Dois aspectos interessantes que devem ser levados em consideração:

1. As variáveis *$consulta* e *$resultados*, recebem os seus respectivos valores em forma de *array*. Por isso conseguimos imprimir seu valor usando o código:

```
$resultados[0];
```

O índice zero equivale ao campo *ID*, e assim por diante;

2. Outro fator muito importante é que conseguimos facilmente criar código HTML usando o *print* do PHP. Veja que estamos "desenhando" a tela usando *print* e as *tags* de tabela, tais como *<tr>* e *<td>*.

Veja o resultado na **Figura 11.** Nela podemos ver todos os usuários do banco sendo listados.

Considerações finais

O PHP é uma linguagem sensacional, porém a produtividade é baixa considerando que programamos todo o site utilizando o Bloco de Notas. Certamente essa não é a única maneira de fazê-lo. Existem editores de HTML que aceitam o código PHP e ajudam, ao menos, colorindo o código-fonte ("highlight").

Em se tratando de IDE de desenvolvimento, o Delphi for PHP é a primeira IDE totalmente voltada ao desenvolvimento em PHP. Sua versão 2.0 acaba de ser lançada, provando que a CodeGear está empenhada em investir na ferramenta.

Nessa aplicação poderíamos ter criado uma pasta imagens e outra *admin* para isolar o arquivo referente à listagem de usuários e também para termos as imagens em um só local e organizado.

Outra preocupação inerente à programação PHP com Banco de Dados diz respeito a serviços de hospedagem que utilizem o Firebird. No Brasil os bancos de dados na internet mais populares são o *SQL Server*, *PostgreSQL* e *MySQL*, portanto pesquise bastante antes de fechar qualquer parceira ou mesmo serviço de hospedagem.

Conclusão

O PHP realmente é uma linguagem poderosa e muito robusta. Há muito ainda o que falar dela e com certeza não faltarão oportunidades. O recado que deixo aqui é para que você estude a fundo todas as suas funcionalidades e procure exercitar o aprendizado. Só assim é possível usufruir de 100% do que a linguagem oferece.

O que vimos aqui foi apenas a ponta do iceberg, cabe a você aprofundar-se mais e mais na linguagem. Um forte abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

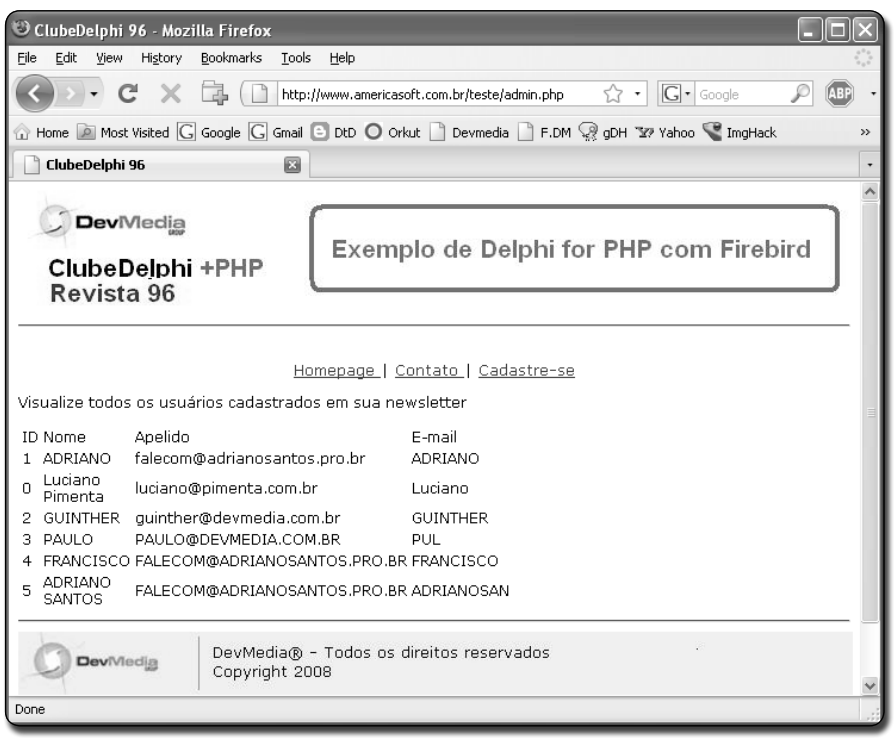


Figura 11. Resultado final da listagens de usuários

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Iniciando o uso de Web Services no .NET

Construa e utilize na prática Web Services em suas aplicações ASP.NET

Nesse artigo veremos

• Criação de um Web Services na prática;
• Utilização do Web Service criado.

Qual a finalidade

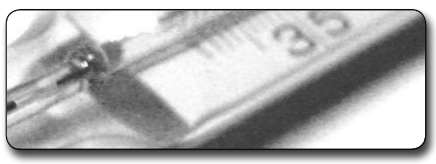
• Os Web Services podem ser muito úteis, pois podem ser facilmente acessados e podem prover acesso ao banco de dados ou funções específicas de maneira bastante transparente.

Quais situações utilizam esses recursos?

• Uma situação bastante interessante para a utilização de um Web Service é quando necessitamos que nossa aplicação seja acessada de diferentes dispositivos e meios de acesso, ex.: Um banco de dados pode ser lido pelo Desktop, pela internet, por dispositivos móveis como PocketPC's, Palm's e celulares, além é claro, de poder estar disponível para aplicações escritas em Delphi, PHP, ASP.NET, entre outras.

Acompanhando a evolução das ferramentas de desenvolvimento e da internet, os *Web Services* também evoluíram de plataforma. Faremos aqui a criação de um *Web Service* baseado em ASP.NET, ao invés de Win32 (visto na edição 95). A criação do serviço utilizando outra plataforma não irá alterar a compatibilidade entre as demais plataformas. Sua acessibilidade permanecerá a mesma entre Win32 e .NET Framework, bem como entre as mais diversas ferramentas de desenvolvimento utilizadas. Para criação do WS ("Web Service"), utilizaremos o Borland Developer Studio 2006. Desenvolveremos um serviço ("Server") e uma aplicação cliente, onde iremos simular um conversor de temperaturas. As regras de conversão ficarão centralizadas na camada servidora do serviço através de funções com retorno de valores e

Maikel Marcelo Scheid
maikelscheid@gmail.com
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.



as referências serão feitas por meio de uma aplicação cliente baseada na plataforma ASP.NET.

Criando ASP.NET Web Service

Acesse o menu *File>New>Other>Delphi for .NET Projects>ASP.NET Web Service Application* (**Figura 1**). Em seguida dê o nome de "Conversor" na janela *New ASP.NET Web Service Application* (**Figura 2**). Ainda na caixa de diálogo, observe que nosso *WS* será criado no diretório *"c:\Inetpub\wwwroot\Conversor"* e que o *Server* padrão será o *Microsoft Internet Information Services (IIS)*, servidor Web da MS que acompanha o Windows. Mantenha as

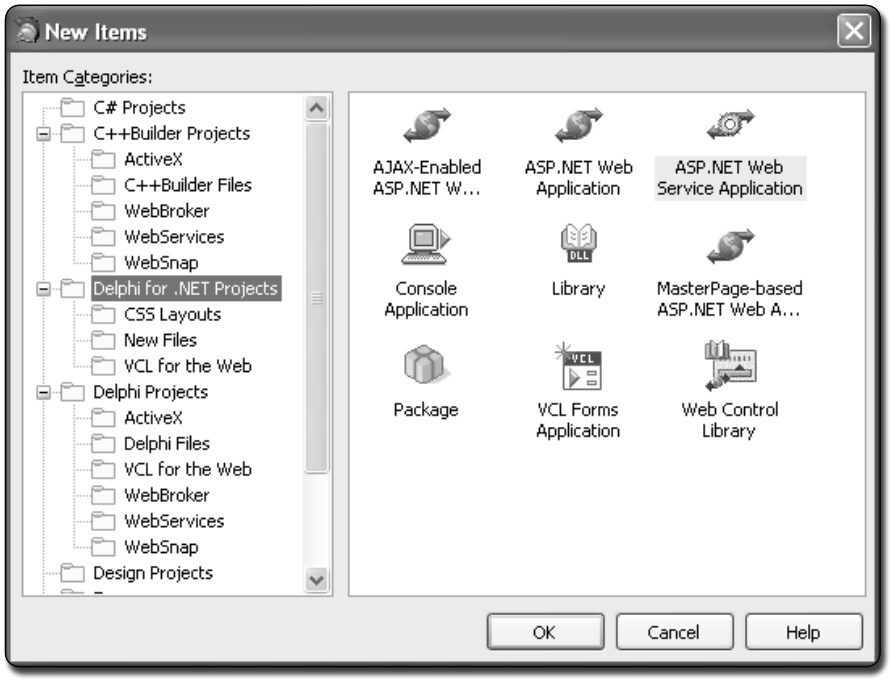


Figura 1. Criando ASP.NET Web Service Application

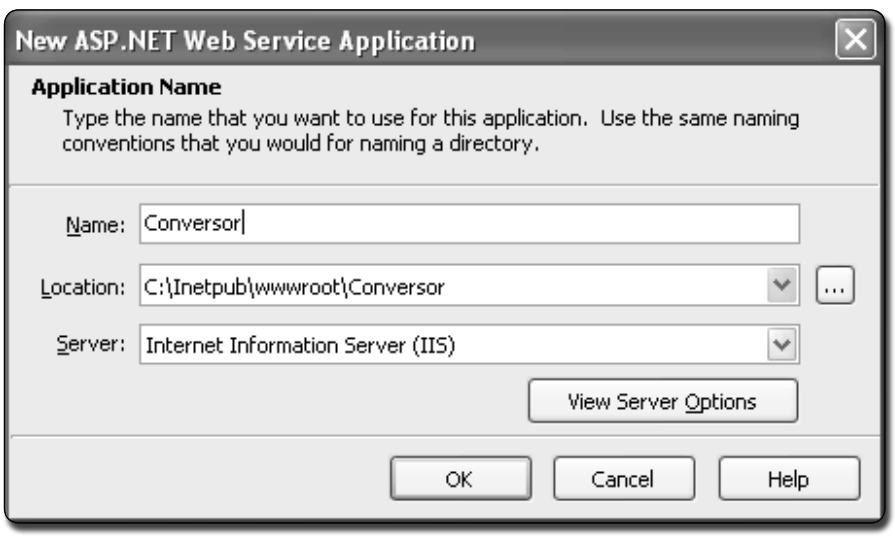


Figura 2. Configuração do nome e diretório virtual para o serviço

Nota do DevMan

Web service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os Web services são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML.

Para as empresas, os Web services podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de logística. Toda e qualquer comunicação entre sistemas passa a ser dinâmica e principalmente segura, pois não há intervenção humana.

Para a representação e estruturação dos dados nas mensagens recebidas/enviadas é utilizado o XML (eXtensible Markup Language). As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol) (baseado em XML). Os serviços (operações, mensagens, parâmetros, etc.) são descritos usando a linguagem WSDL (Web Services Description Language).

Nota do DevMan

Um método assinalado com o atributo [Web Method] indica que o mesmo é um método de Web Service e pode ser invocado remotamente. O suporte atributos foi adicionando no compilador do Delphi for .NET, ainda no preview que acompanha o D7, para suportar diretivas do .NET Framework.

Listagem 1. Declarando as funções de conversão

```
…
public
  [WebMethod]
  function ConverterGraus(De,Para : String;
    Valor : Double) : Double;
end;
…
```

configurações como estão e confirme a caixa de diálogo. Após a criação do serviço o Delphi irá direcioná-lo para o *Layout* de criação e codificação das regras a serem utilizadas.

Criado o projeto, expanda o item *WebService1.asmx* no *Project Manager* e clique duas vezes sobre o arquivo *WebService1.pas*. Localize a seção *public* da classe, onde já se encontra registrada uma *procedure* chamada "HelloWord" que nada mais é do que um exemplo criado pelo IDE. Faremos as declarações da função que iremos utilizar na conversão, declarando conforme modelo da **Listagem 1.**

Observe nas declarações que declaramos a função logo abaixo da referência do *WebMethod* indicando que será um método público e disponível para consultas na *Web*. Cada função irá receber como parâmetros de entrada o formato de valor de origem e valor de saída desejado, bem como o valor a ser convertido. Após a declaração da função utilize as teclas *Ctrl+Shift+C* para que o Delphi faça crie o cabeçalho do método, ao qual deverá ser adicionado o código da **Listagem 2** que encontra-se comentado de acordo com a ação a ser executada.

Na codificação da função anterior estamos, antes de tudo, avaliando quais os parâmetros de conversão fornecidos pelo usuário para que seja realizado o cálculo adequado entre as temperaturas. Após atender a condição correta do cálculo, para cada situação estamos atribuindo um valor a uma variável criada logo no início da codificação da função. Realizado o cálculo, o valor armazenado na variável será devolvido ao resultado da *function*, valor este que será o retorno ao usuário durante a chamada da função no *Web Service*.

Finalizada a codificação, poderemos fazer um teste relacionado ao funcionamento dos cálculos da nossa aplicação. O teste pode ser feito executando o *Web Service* através do atalho *Ctrl+Shift+F9* ou utilizando apenas o *Run*. Será criada uma página *web* para *debug* e teste do serviço (**Figura 3**), onde haverá um atalho para o nome da função criada. Clicando no link a página será redirecionada automaticamente a um formulário para preenchimento dos dados. Para testar basta informar os dados dos campos *De* e *Para* referentes temperatura e o campo *Valor*. O retorno da função é feito em formato XML (*eXtensible Markup Language*) (**Figura 4**).

Conferidos os resultados durante os testes, finalizamos a etapa do desenvolvimento da versão *servidora* para a construção do *Web Service*.

Listagem 2. Conversão de Temperaturas

```
function TWebService1.ConverterGraus(De, Para:
  String; Valor: Double): Double;
var
  aValue : Double;
begin
  if ((De = 'C') and (Para = 'F')) then
  begin
    { Graus Celsius para Fahrenheit }
    { Fº = (Cº * 1.8) + 32}
    aValue := (Valor*1.8)+32;
    Result := aValue;
  end;
  if ((De = 'C') and (Para = 'K')) then
  begin
    { Graus Celsius para Kelvin }
    {Kº = ºC + 273}
    aValue := (Valor + 273);
    Result := aValue;
  end;
  if ((De = 'F') and (Para = 'C')) then
  begin
    { Graus Fahrenheit para Celsius }
    {Cº =(Fº-32)/1,8}
    aValue := (Valor-32)/1.8;
    Result := aValue;
  end;
  if ((De = 'F') and (Para = 'K')) then
  begin
    { Graus Fahrenheit para Kelvin }
    {Cº=(Fº-32)/1,8 - Convertemos primeiro para Cº}
    {Kº = ºC + 273 - De Cº vamos converter para Kº}
    aValue := (Valor-32)/1.8;
    aValue := aValue + 273;
    Result := aValue;
  end;
  if ((De = 'K') and (Para = 'F')) then
  begin
    { Graus Kelvin para Fahrenheit }
    {ºC = Kº - 273 - Convertemos primeiro Kº para Cº}
    { Fº = (Cº * 1.8) + 32}
    aValue := Valor - 273;
    aValue := (aValue*1.8)+32;
    Result := aValue;
  end;
  if ((De = 'K') and (Para = 'C')) then
  begin
    { Graus Kelvin para Celsius }
    {ºC = K - 273}
    aValue := Valor - 273;
    Result := aValue;
  end;
end;
```

Criando a aplicação cliente

Inicie uma nova aplicação usando o menu *File>New>Other>Delphi for .NET Projects>ASP.NET Web Application* atribuindo a ela o nome de "ClienteConversor". Mantenha as demais configurações *default*. Na página *WebForm1.aspx* adicione ao corpo da página dois componentes *DropDownList* ("ddlDe" e "ddlPara") da paleta *Web Controls*.

Selecione o controle *ddlDe* e então clique duas vezes em sua propriedade *Items*. Clique em *Add* para criar um novo item. Em *Text* digite "Cº" e em *Value* "C". Repita os mesmos passos para as demais temperaturas assim como visto na **Figura 5**. Ao final da configuração, refaça a inclusão dos mesmos itens no *DropDownList ddlPara*.

> **Nota**: Cuidado para não utilizar caracteres especiais na propriedade *Value* onde os valores deverão estar de acordo com as verificações feitas anteriormente durante a criação da função de conversão na camada servidor do *Web Service*.

Adicione também ao formulário dois campos *TextBox*("txtValor" e "txtResultado") que serão utilizados para receber o valor a ser convertido e o resultado da conversão, respectivamente. Por fim inclua um *Button*("BrnConverter") que fará a chamada da função de conversão.

Antes de codificarmos o botão de conversão, precisamos importar o caminho de referência para o servidor do *Web Service*. Localize *Project Manager* a DLL da criação do seu projeto e com um clique do botão direito do mouse selecione a opção *Add Web Reference*. Um *Wizard* será aberto para a adição do caminho ao arquivo principal do serviço. Nesse ponto devemos informar o caminho do WebService criado anteriormente que por *default* será *http://localhost/Conversor/WebService1.asmx*. Digite o endereço na URL do *Wizard* e pressione *Enter*. Note que ao ser encontrada a página, o *Wizard* habilitará automaticamente o botão *Add Reference*, então basta clicar no mesmo que a importação da referência será efetuada (**Figura 6**).

Ao importar o arquivo, observe que no *Project Manager* surgirá um novo item com as referências externas adicionadas ao projeto (**Figura 7).** A partir de agora, podemos simplesmente realizar a chamada da função embutida no WS adicionando o código da **Listagem 3** ao evento *Click* do botão *btnConverter*. Adicione também referência da página para a *Unit* do *Web Service* através do atalho *Alt+F11;*

Na codificação do *btnConverter*, criamos antes de tudo uma variável que receberá o objeto instanciado pelo *WS*, e as demais variáveis para receber os valores a serem passados para o serviço bem como o resultado da conversão.

A conversão é simples: primeiro instanciamos o serviço usando a variável *converter* que recebe por atribuição o próprio WS. Em seguida verificamos se o valor selecionado em ambos *DropDownLists* são diferentes, para evitar que o usuário tente converter um valor nele mesmo. Caso a validação seja positiva guardamos os valores selecionados e o grau digitado em *txtValor*. Para que seja calculada a conversão, passamos o conteúdo das variáveis para o método *ConverterGraus*, presente em nosso WS. Seu resultado é mostrado no *txtResultado*. Em caso negativo, informamos ao usuário que os formatos de temperatura devem ser diferentes (**Figura 8**).

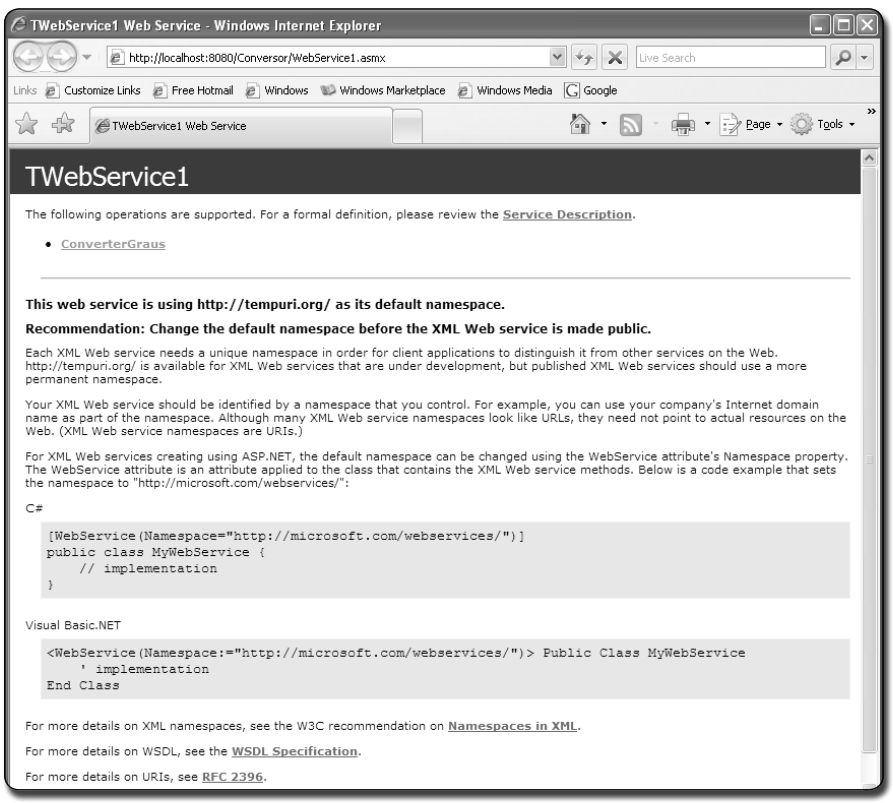


Figura 3. Executando o serviço

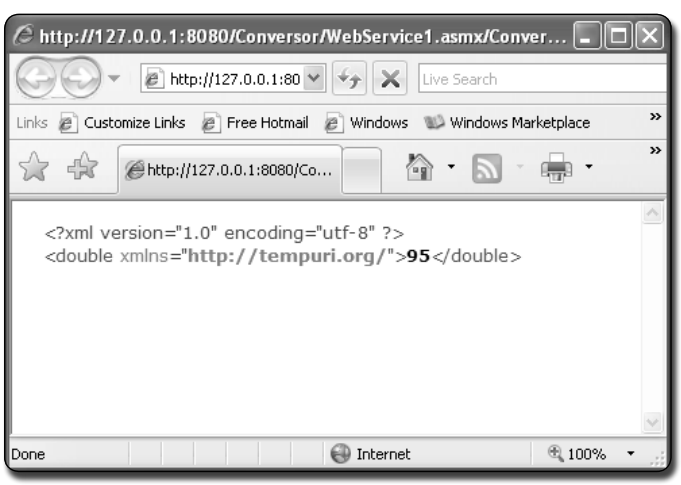


Figura 4. XML com resultado da conversão de temperaturas

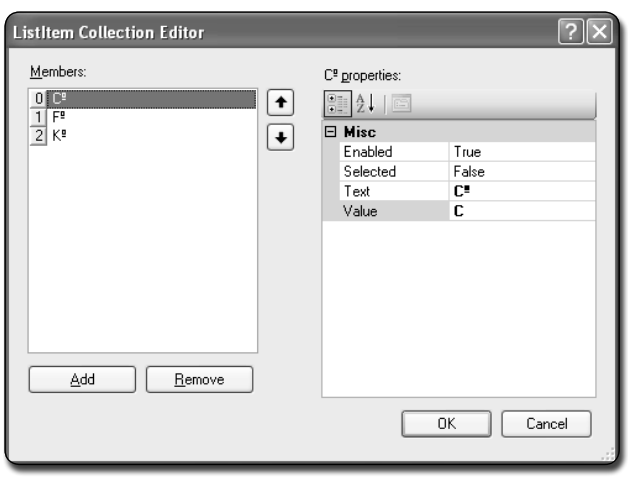


Figura 5. Criando propriedades dos itens de conversão

Conclusão

Vimos, através deste exemplo, uma forma prática de criar serviço também sobre a plataforma .NET. Faça também a importação deste mesmo serviço em suas aplicações Win32 implementando outros recursos, baseando-se no meu artigo da edição 95. Claro que o desenvolvimento de Web Services não pára por aqui. Com essa tecnologia podemos disponibilizar nossas aplicações para diversas plataformas, incluindo dispositivos móveis, tais como: PocketPC's, SmartPhones, aparelhos de GPS e Palm's. Abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Figura 6. Adicionando a referência do serviço

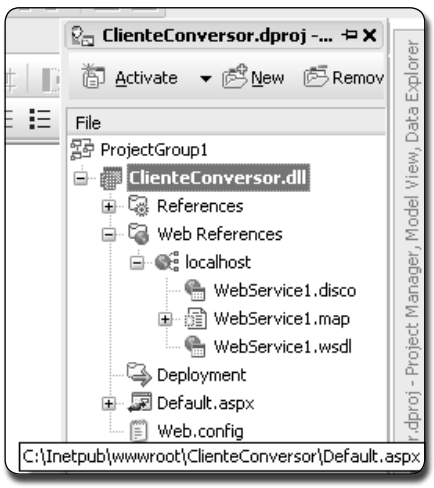


Figura 7. Referência adicionada ao projeto

Listagem 3. Evento Click do botão de conversão

```
procedure TDefault.btnConverter_Click(sender: TObject; e: System.EventArgs);
var
  converter : TWebService1;
  De: String;
  Para : String;
  Valor : Double;
  Resultado : Double;
begin
  converter := TWebService1.Create;
  if ddlDe.SelectedValue <> ddlPara.SelectedValue then
  begin
    De := ddlDe.SelectedValue;
    Para := ddlPara.SelectedValue;
    Valor := Convert.ToDouble(TxtValor.Text);
    Resultado := converter.ConverterGraus(De, Para, Valor);
    txtResultado.Text := Resultado.ToString;
  end
  else
    Response.Write('Não é possível converter para o mesmo formato!');
end;
```

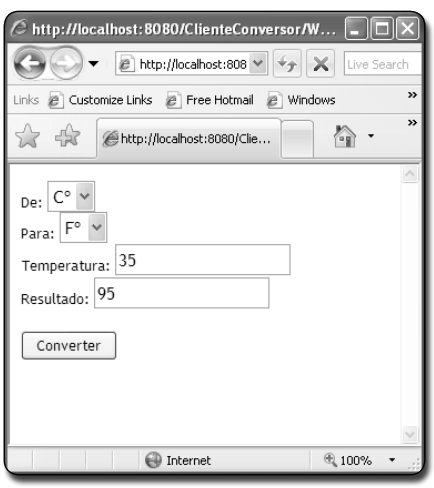
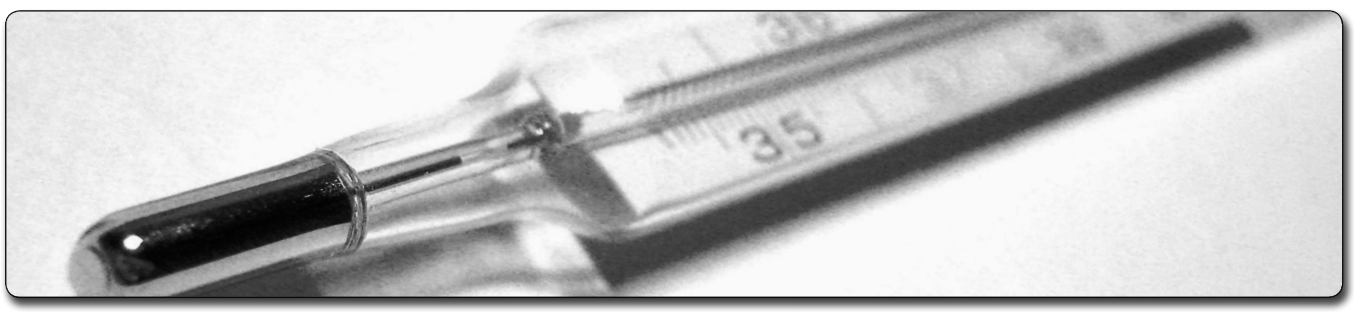


Figura 8. Conversão de temperaturas utilizando Web Services



A EDIÇÃO QUE VOCÊ PRECISA ESTÁ ESGOTADA?

SEUS PROBLEMAS ACABARAM!!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

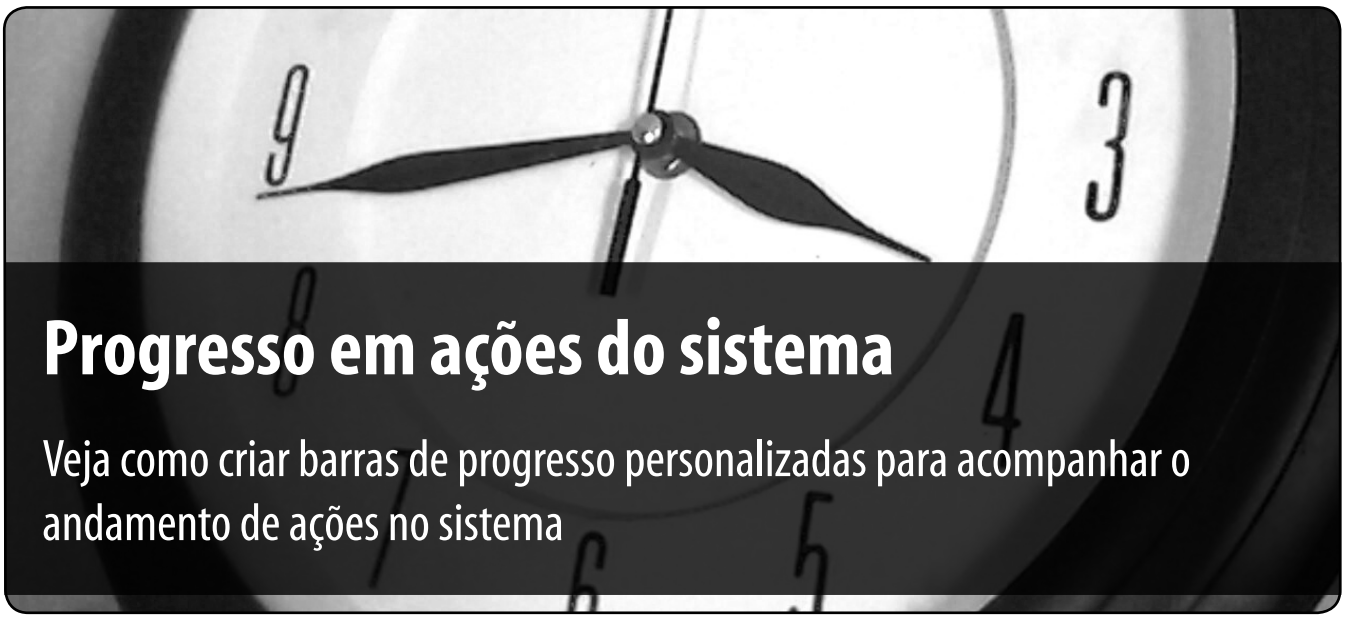
Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

Nesta seção você encontra artigos para iniciantes na linguagem Delphi



Progresso em ações do sistema

Veja como criar barras de progresso personalizadas para acompanhar o andamento de ações no sistema

Maikel Marcelo Scheid
maikelscheid@gmail.com
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

A curiosidade do usuário em saber e acompanhar o andamento dos processos no sistema está bastante visível nas solicitações que são feitas ao desenvolvedor antes de iniciar o sistema. Os progressos variam desde uma ação demorada de exportação/importação de registros, execução de músicas, upload/download de arquivos da Web entre outras situações em que podemos empregar o uso de componentes do Delphi para permitir esse acompanhamento ao usuário.

Imagine seu sistema exportando um relatório em PDF e ao mesmo tempo fazendo upload para um servidor de FTP da internet e o usuário estar acompanhando a porcentagem exata referente ao total do trabalho concluído, ou então, crie seu próprio *player* de áudio e mostre o progresso das músicas para acompanhamento do usuário.

Baseado nestes desafios comentados, veremos neste artigo uma introdução básica sobre a utilização dos componentes de progresso (*TrackBar*, *ProgressBar* e *Gauge*). Criaremos um exemplo onde você fará downloads de arquivos dos sites da internet e irá acompanhar através da sua aplicação o percentual do trabalho concluído no *Gauge*, também criaremos um pequeno *player* de áudio utilizando o componente *TrackBar* para exibir o andamento da música. Por final simularemos a exportação dos dados de um componente *Table* conectado ao *alias DBDEMOS* e atualizaremos o progresso do trabalho em um componente *ProgressBar*.

Download de arquivos com barra de progresso

Utilizando o Delphi 7 para desenvolver nosso exemplo, crie uma nova aplicação no menu *File|New>Application*. Altere o nome do formulário principal para "frmDownload" e o *Caption* para "Pregresso em Downloads". Salve a *unit* do formulário em um diretório do seu computador atribuindo-lhe

o nome "uProgress.pas" e logo após salve o projeto como "prjProgress.dpr". Arraste da paleta *Standard* um componente *Label*("lblDescricao") e altere o *Caption* para "URL para Download". Arraste da mesma paleta um componente *Edit*("edtURL") e um componente *Button*("btnDownload"), removendo o texto do *edtURL* e alterando a propriedade *Caption* do *btnDownload* para "Download".

Para esta aplicação estamos construindo um *layout* para que possamos fazer download de arquivos de servidores da internet. Nossa intenção será fornecer o *link* da aplicação que desejamos "baixar" no *edtURL* do aplicativo e o mesmo realizar o *download* para nós, exibindo o progresso do serviço. Quando falamos em *download* e utilização de componentes para tal tarefa lembramos logo dos componentes das paletas *Indy* que desempenham muito bem estas atividades.

Posicionado na paleta *Indy Clients* arraste para o formulário um componente *IdHTTP*("IdHTTP1") que fará as conexões de busca com a internet. Para exibir o progresso do *download*, utilizaremos neste exemplo o componente *Gauge*("gauge1") da paleta *Samples*, que leva uma pequena vantagem relacionando aos demais componentes de progressos relacionados anteriormente, no sentido de qualidade das informações. O *Gauge* além de mostrar o preenchimento da barra de progresso, exibe de forma numérica o percentual exato do trabalho concluído.

Adicionado então o *Gauge* ao formulário, iremos ainda da paleta *Standard* adicionar um novo *Label*("LblDownload") e altere o *Caption* para "Aguardando...", onde irei demonstrar o progresso do *download* baseado no tamanho do arquivos em *KB* ao qual estamos fazendo o *download* e a quantidade de *KB* que já foram salvos no disco do computador. Por fim adicione ainda da paleta *Dialogs* um componente *SaveDialog*("SaveDialog1") que utilizaremos para solicitar ao usuário um local onde o mesmo deseja realizar o download do arquivo. Organize todos os componentes de forma semelhante à aplicação da **Figura 1**, onde daremos início logo mais à codificação dos eventos.

Até o momento havíamos só falado no *layout* do sistema, onde passaremos agora as codificações a fim de deixar o mesmo funcional. Os códigos a serem utilizados serão simples e de fácil entendimento. O primeiro passo será a criação de uma *procedure* a qual iremos chamar no momento em que realizarmos o *download*. Declare entre o método *Private* da *unit* a *procedure* a seguir e utilizando as teclas *Ctrl+Shift+C* crie sua implementação:

```
procedure DownloadFile(Origem, Destino:
String);
```

Adicione ao corpo do procedimento o código da **Listagem 1,** o qual encontra-se comentado de acordo com as ações que serão executadas em cada uma das linhas de código, desde a atualização dos componentes da aplicação, criação de uma variável de *Streaming* de memória para armazenar o arquivo enquanto é realizado o *download* e logo mais a conexão do componente *IdHTTP1* ao arquivo solicitado na URL por meio do seu método *Get*, salvando então a solicitação. Por final temos ainda o tratamento dos erros caso ocorra algum e logo mais a limpeza dos componentes da memória.

A chamada a este procedimento será realizada no evento *OnClick* do *btnDownload* onde o código da **Listagem 2** deverá ser adicionado. Nesta codificação iniciamos com uma verificação do preenchimento do *edtURL* com algum valor de URL para que possa ser dada a seqüência do *download*. Encontrado o texto da *URL*, criamos uma pequena rotina que irá passar entre todos os caracteres do texto, pesquisando pelo símbolo "/" a fim de encontrar a ultima ocorrência válida e obter todo o texto que está a sua direita, mantendo-o como nome sugestivo para salvar o arquivo em disco. Este nome é sugerido ao componente *SaveDialog1* executado na linha seguinte, solicitando um local para armazena-

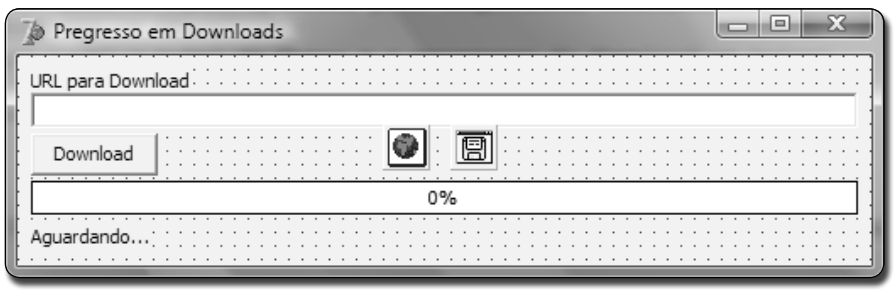


Figura 1. Layout da estrutura para exibir progresso de downloads

Listagem 1. Procedure para download de arquivo

```
procedure TfrmDownload.DownloadFile(Origem, Destino: String);
var
  fsFromFile: TFileStream;
begin
  {atualiza e processa as ações ocorrendo na aplicação}
  Application.ProcessMessages;
  {cria um stream temporário para o download}
  fsFromFile := TFileStream.Create(Destino, fmCreate );
  try
    try
      {obtém o arquivo solicitado na URL e salva-o em disco}
      idHTTP1.Get(Origem, fsFromFile );
    except
      on Error: Exception do
      begin
        {tratamento dos erros quando houver}
        ShowMessage('O seguinte erro ocorreu durante o download:'+#13+Error.Message );
        Raise;
      end;
    end;
  finally
    {limpa a variável da memória e desconecta o componente}
    fsFromFile.Free;
    idHTTP1.Disconnect
  end;
end;
```

Listagem 2. Iniciando o download do arquivo

```
procedure TfrmDownload.btnDownloadClick(Sender: TObject);
var
  vArquivo: TFileStream;
  vFileName : String;
  I: Integer;

begin
  {somente inicia se houver uma URL no campo}

  if Length(edtURL.Text) > 0 then
  begin
    {Passando a URL para a variável a fim de obter o nome do arquivo}
    vFileName := edtURL.Text;

    {inicia laço e pesquisa pela última barra para obter filename}
    for I := 0 to Length(vFileName) - 1 do
    begin
      {se encontrar uma barra deleta os caracteres que antecedem a mesma.
      Se chegar na última barra permanece apenas o nome do arquivo}
      if Pos('/',vFileName) > 0 then
      Delete(vFileName,1,Pos('/',vFileName));
    end;

    {abre o diálogo já preenchido com o nome e solicita local de armazenamento do download}
    SaveDialog1.FileName := vFileName;

    if SaveDialog1.Execute then
    begin
      {inicia o download do arquivo}
      DownloadFile(edtURL.Text,SaveDialog1.FileName);
    end;

  end
  else
  MessageDlg('Informe a URL do Download!',mtWarning,[mbOK],0);
end;
```

Listagem 3. Eventos durante download de arquivos

```
procedure TfrmDownload.IdHTTP1WorkBegin(ASender: TObject;
          AWorkMode: TWorkMode; AWorkCountMax: Integer);
begin

  {limpa a barra de progresso}
  Gauge1.Progress := 0;

  {define o tamanho máximo para o Gauge}
  Gauge1.MaxValue := AWorkCountMax;
end;

procedure TfrmDownload.IdHTTP1Work(ASender: TObject;
          AWorkMode: TWorkMode;AWorkCount: Integer);
begin
  {a cada conjunto de bytes trazidos pelo componente, incrementamos o Gauge}
  Gauge1.Progress := AWorkCount;

  {mostra no label o total de bytes e o progresso do download em bytes}
  LblDownload.Caption  := 'Baixando '+FormatFloat('#0.00',
    (AWorkCount/1024)/1024)+'MB de '+FormatFloat('#0.00',(Gauge1.MaxValue/1024)/1024)+'MB';

  {reescreve os valores no formulário}
  Repaint;
end;

procedure TfrmDownload.IdHTTP1WorkEnd(ASender: TObject; AWorkMode: TWorkMode);
begin
  {conclusão do trabalho, fim do download}
  LblDownload.Caption := 'Download concluído!';
end;
```

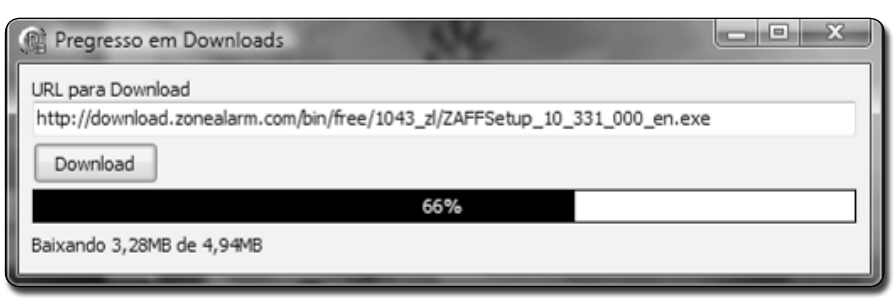


Figura 2. Download de arquivo

mento do arquivo. Definido o destino, chama-se a procedimento responsável pelo download que criamos há pouco e dá-se início a atividade.

Bom, até o momento falamos e configuramos somente os códigos para *download*, sem nada de progresso para os mesmos, mas que veremos agora, com a codificação de três eventos onde a identificação de nomes de cada um deles já indica sua atividade. Selecionando o componente *IdHTTP1*, note a existência dos eventos *OnBeginWork* (quando o serviço é iniciado), *OnWork* (durante o serviço) e *OnWorkEnd* (quando o serviço é concluído), que codificaremos de acordo com a **Listagem 3**, fazendo todos os controles de progresso necessários.

Logo na inicialização da solicitação de download, o primeiro evento a ser ativado será o evento *WorkBegin*, onde posicionamos o componente *Gauge1* na posição zero e definimos qual será seu valor máximo, atribuindo a ele o tamanho em *Bytes* do arquivo da solicitação. Logo mais, no evento *Work*, que significa durante o trabalho de download, mantemos atualizados os valores obtidos de acordo com o serviço do componente, atualizamos no *Gauge1* o percentual de *Bytes* já transferidos, que serão convertidos no progresso do componente. Também atualizamos o tamanho parcial e total de Bytes, transformados agora em *KB* do arquivo a ser salvo. Por final, ao concluir o download, alteramos o *Caption* do *Label* para *concluído* indicando o fim do *download*.

Realizadas todas as codificações, execute seu aplicativo e obtenha um *link* de internet para que possa realizar o *download* e observar o progresso no funcionamento do seu sistema (**Figura 2**). Durante o funcionamento, a velocidade do progresso irá variar muito de acordo com o tamanho do arquivo solicitado e também da velocidade de conexão com a internet, claro.

Progresso de andamento em músicas

Para visualizarmos a utilização do componente *TrackBar*, criaremos um exemplo de um pequeno *player* de música, onde ao executarmos o áudio o componente

de progresso irá avançando de acordo com o andamento de execução. No menu *File|New>Application* crie um novo projeto, alterando o nome do formulário principal para "frmPMusica" e seu *Caption* para "Progress em músicas". Salve a *unit* do formulário com "uPMusicas.pas" e o projeto como "prjPMusicas.dpr".

Para executar os arquivos de áudio, adicione da paleta *System* um componente *MediaPlayer*("player"). Para localizar o arquivo a ser executado adicione da paleta *Standard* um *Button*("btnBuscar") alterando o *Caption* para "Buscar" e da paleta *Dialogs* um *OpenDialog*("OpenDialog1"). Para a exibição do progresso e atualização das informações no decorrer da execução da música precisaremos adicionar da paleta *System* um componente *Timer*("Timer1"), *Label*("Label1") da paleta *Standard* e o componente de progresso *ProgressBar*("ProgressBar1") da paleta *Win32*. Organize os componentes de forma semelhante à demonstração da **Figura 3.**

A codificação deste exemplo também é de forma bastante simples e de fácil compreensão, consistindo basicamente em dois procedimentos, um para buscar e abrir o arquivo de áudio e outro no *timer* para atualizar o tempo de duração do áudio. Adicione no evento *OnClick* do *btnBuscar* o código da **Listagem 4** que está comentado de acordo com a ação de cada linha. Antes de adicionar os códigos ao botão, vá até o código do formulário e declare no método *Private* as seguintes variáveis:

```
public
  a,b:TTime;//Variáveis para controle de tempo
end;
```

Ao buscar um arquivo de áudio, a posição inicial da barra de progresso é posicionado no valor zero, habilitando-se em seguida o *player* de áudio e executando o diálogo para a pesquisa dos arquivos. Ao selecionar o arquivo, o mesmo é repassado ao *player* que inicializa sua execução, jogando valores de tempo de duração máxima e posição da execução ao componente do progresso. A variável *b* recebe a hora em que o mesmo foi executado e será utilizada para exibir a duração da execução do mesmo.

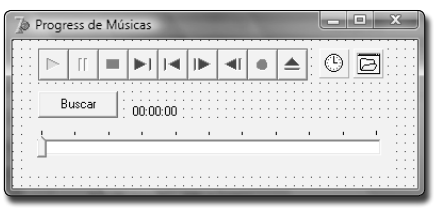


Figura 3. Player de Música com progresso

Os códigos responsáveis por manter atualizados o progresso e o tempo da execução serão adicionados no evento *OnTimer* do *Timer1*, onde deverá ser adicionado o código da **Listagem 5**, na qual verificamos o estado do *player*.

Concluída a codificação dos componentes, execute sua aplicação e localize um arquivo de áudio para execução, observando o comportamento dos mecanismos de progressos configurados (**Figura 4**).

Progresso com operações em DataSets

Imagine seu sistema com uma rotina que exige a exportação de registros carregados em um componente *DataSet*, que ao clicar em "Iniciar Exportação" o usuário possa acompanhar o andamento da exportação em barras de progresso semelhantes as barras do Windows XP, deixando seu sistema com uma aparência mais profissional. Semelhante como descrito, faremos uma simulação utilizando *DataSets* onde iremos percorrer todos seus registros e mostrar a posição de localização dos ponteiros em um componente *ProgressBar*.

No menu *File|New>Application* crie uma aplicação, altere o nome do formulário principal para "frmPTable" e o *Caption* para "Progress com Table". Salve a *unit* do formulário como "uPTable.pas" e o projeto como "prjPTable.dpr". Adicione da paleta BDE um componente *Table*("Table1") e na propriedade *DatabaseName* selecione o *alias* "DBDEMOS" e em *TableName* selecione a tabela "employee.db" alterando sua propriedade *Active* para *True*. Arraste da paleta *Data Access* um componente *DataSource*("DataSource1") e relacione a propriedade *DataSet* ao *Table1*. Arraste

Listagem 4. Buscando o arquivo de áudio e iniciando execução

```
procedure TfrmPMusica.btnBuscarClick(Sender: TObject);
begin
  //posição inicial do progresso
  TrackBar1.Position:=0;
  //player é habilitado
  player.Enabled:=False;
  //arquivo de áudio é passado ao player
  if OpenDialog1.Execute then
  begin
     //player inicia a execução do áudio
     player.Close;
     player.FileName:=OpenDialog1.FileName;
     player.Open;
     player.Play;
     player.Enabled:=TRue;
     TrackBar1.Max:=player.Length;
     TrackBar1.Position:=player.Position;
  //Hora do início da execução sendo armazenado
     b:=Time;
   end;
end;
```

Listagem 5. Atualizando intervalos de tempo e progresso

```
procedure TfrmPMusica.Timer1Timer(Sender: TObject);
begin
  //Quando o player está parado, tempo de música é zerado
  if player.Enabled = False then
  Begin
    a:=0;
    Label1.Caption:='00:00:00';
    TrackBar1.Position := 0;
  End;
  //Quando player em execução o progresso é atualizado
  if player.Enabled then
  begin
    //a recebe a hora atual
    a:=Time;
    //Label1 recebe o tempo atual menos tempo incial
    Label1.Caption:=TimeToStr(a-b);
    //TrackBar1 recebe a posição igual ao Player
    TrackBar1.Position:=player.Position;
  end;
  //quando a música chega ao fim, o player é parado
  if player.Position = TrackBar1.Max then
  player.Enabled := false;
end;
```

também da paleta *Data Controls* um *DBGrid*("DBGrid1") e ligue-o através da propriedade *DataSource* ao *DataSource1* recém configurado, fazendo com que os registros já sejam exibidos.

Para demonstrar o progresso de interação com os registros deste componente, arraste para o formulário da paleta *Win32* um componente *ProgressBar*("ProgressBar1") e da paleta *Standard* e um *Button*("Button1"), ao qual iremos adicionar um código que irá percorrer todos os registros carregados e exibir o andamento na barra de progresso. Altere o *Caption* do botão para "Simular Transferência";

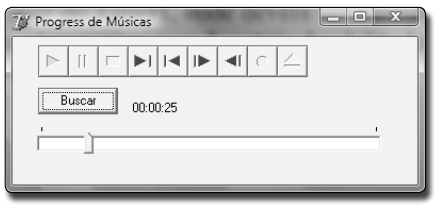


Figura 4. Progresso da execução do áudio

No evento *OnClick* do botão adicione o código da **Listagem 6**, que irá definir o valor inicial para o componente, o número de registros encontrados será utilizado como valor máximo que o progresso irá alcançar e logo após posicionamos os registros da *Table1* na primeira linha. A seguir iniciamos um laço que irá percorrer todos os registros até seu final, atualizando o posicionamento do componente de progresso. Lembrando que esta será apenas um simulação de exportação, que não será configurada neste artigo devido a quantidade de espaço que levaríamos para a configuração. Para poder observar melhor o progresso da barra, adicionamos uma pequena pausa de 200 milissegundos após cada registro percorrido.

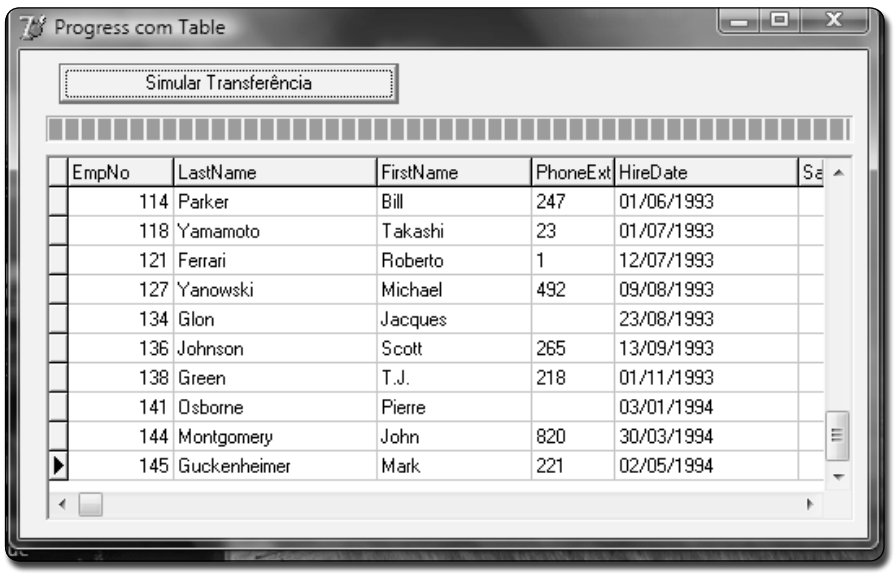


Figura 5. Progresso na simulação de exportação

Listagem 6. Simulando uma exportação

```
procedure TfrmPTable.Button1Click(Sender: TObject);
begin
  //posiciona a barra no inicio
  ProgressBar1.Min := 0;
  //define o valor máximo
  ProgressBar1.Max := Table1.RecordCount;
  //posiciona o Table1 no primeiro registro
  Table1.First;
  //Laço percorre todos os registros
  while not Table1.Eof do
  begin
    //atualização da barra de acordo com posição do Table1
    ProgressBar1.Position := Table1.RecNo;
    //pausa de 200 milissegundos
    sleep(200);
    //avança para o próximo registro
    Table1.Next;
  end;
end;
```

Realizada a codificação, execute sua aplicação e clique sobre o botão que dará início a simulação de exportação, observado o progresso da barra enquanto o componente *Table1* irá percorrendo cada um dos registros (**Figura 5**).

Conclusão

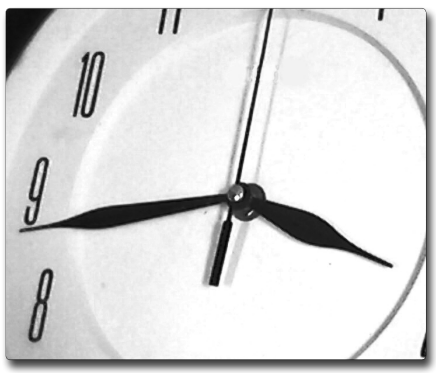
No decorrer deste artigo vimos três situações diferentes para utilização de componentes que exibem o progresso de ações, com as quais poderá dar novas funcionalidades ao seu sistema, construir procedimentos em que o usuário possa acompanhar o atual andamento do processo que está ocorrendo. As demonstrações aqui utilizadas foram bem simples, onde você poderá criar e implementar seus próprios procedimentos. Configure progresso em download de endereços de FTP ou mesmo *upload* de arquivos para um servidor FTP, onde os códigos serão muito semelhantes aos já apresentados. No progresso de música, implemente para que o usuário possa adiantar ou retardar a execução das mesmas, deixando-o interagir mais com o *player* e por fim, no último exemplo, faça outras ações que envolvam banco de dados e mostre o progresso das operações. Obrigado e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

PLANO PROFISSIONAL 1

3 domínios independentes
50 GB de transferência
30 GB de e-mails com SSL
1 GB de espaço
Painel de controle
ASP, ASP.NET 3.5 E PHP 5
Access ilimitado
3 bancos MYSQL 5
1 banco SQL Server 2005 Express

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS 1983–2008

SafeNet